DECnet/E
Network Installation Guide

Order No. AA-K714A-TC

January 1982

This manual presents procedures for DECnet/E users and DIGITAL software support specialists to determine whether a node has been built in accordance with defined network generation procedures. The installation procedures also verify whether the communications hardware to be used by the node is operational and whether all supported adjacent nodes in the network are reachable.

Operating System and Version: **RSTS/E V7.1**

Software Version: **DECnet/E V2.0**

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, January 1982

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

Copyright © 1982 Digital Equipment Corporation

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	IAS	SBI
DECnet	TRAX	PDT
DATATRIEVE		

Distributed and Mid-Range Systems Publications typeset this manual using DIGITAL's TMS-11 Text Management System.

310ALL

Contents

Figures

Tables

Preface

Manual Objectives

The *DECnet/E Network Installation Guide* defines procedures for generating, installing and testing a node so that the following criteria are met:

- The permanent parameter file is configured such that it truly reflects the node's status in the network environment.
- The communications hardware to be used by the DECnet node is functional.
- All supported adjacent nodes are reachable.

Intended Audience

This manual is intended for users who are trained in the operation of DECnet/E V2.0. The user should also be trained in the operation of every adjacent node within the network in which the node is to be installed.

Prerequisite Reading

The user of this manual should be thoroughly familiar with the following related documents:

Introduction to DECnet, Order No. AA-J055B-TK

RSTS/E System Generation Manual, Order No. AA-2669E-TC

Update: Order No. AD-2669E-T1

DECnet/E System Manager's Guide, Order No. AA-H505B-TC

DECnet/E Guide to User Utilities, Order No. AA-H504B-TC

DECnet/E Release Notes, Order No. AA-M269A-TC

Structure of Manual

Chapter 1 outlines the procedure for building DECnet/E during the RSTS/E system generation and Chapter 2 describes the procedures for configuring the node. Local node verification procedures are discussed in Chapter 3 and adjacent node verification procedures are discussed in Chapter 4.

Chapter 1
Network Specification during SYSGEN

A DECnet/E system is generated as part of the standard RSTS/E system generation procedure. This chapter describes the relevant steps in the RSTS/E procedure. These steps are as follows:

1. Answer the appropriate configuration questions (see Section 1.2).
2. Mount the DECnet/E distribution medium, as requested by the batch stream during the execution of SYSBAT.
3. Run the BUILD program to build the DECnet/E library programs (see Section 1.3).

After you answer the configuration questions, the SYSBAT batch stream transfers the NSP modules from the DECnet/E distribution medium to the system disk, where they are linked and installed in RSTS/E.

To generate the DECnet/E library, run the RSTS/E BUILD program, specifying the DECnet/E distribution medium and the control file DECNET.CTL as input. Driven by this control file, BUILD compiles the DECnet/E library programs from the distribution medium and installs them on the system disk. It also builds the DECnet/E libraries and file transfer software and then transfers the model control files to the system disk.

After the entire library is built and the system generation procedure is complete, you can tailor the control files to your installation, as discussed in Section 2.4.

1.1 DECnet/E Distribution Kit

DIGITAL supplies the DECnet/E kit on a separate medium from that containing the RSTS/E distribution kit. The DECnet/E medium can be magnetic tape or disk cartridge. The distribution medium includes the following pertinent files, located in account [1,2]:

- **Monitor-level software:**

  NSP.OBJ
  NSPSUB.OBJ
  TRN.OBJ
  NET.OBJ
  SES.OBJ
  XDDINT.OBJ
  XDDVR.OBJ

- **DECnet/E file access software:**
  DAP.OLB (and associated NFT and FAL build files)
- **DECnet/E library programs:**

  NCUCVT.BAS
  NETOFF.BAS
  TLK.BAS
  LSN.BAS
  NET.OLB (and associated build files)
  NPK.OLB (and associated build files)
  NCP.OLB (and associated build files)
  NML.OLB (and associated build files)
  NETACT.BAS
  NCROOT.BAS (root for NETCPY)
  NETFNC.BAS (BASIC–PLUS network function library)
  MIRROR.MAC
- **Control files for building the DECnet/E library:**

  DECNET.CTL
  DAPBLD.CTL
- **Model control and command files for node generation, verification and startup**

  NCPDEF.CTL (permanent parameter file configuration)
  NTESTE.CTL (local node software verification)
  NTESTH.CTL (local node hardware verification)
  NTESTA.CTL (adjacent Phase III node verification)
  NTEST2.CTL (adjacent Phase II node verification)
  DTSDTR.CTL (adjacent Phase II node verification)
  DECNET.CMD (local node startup)

The control files in the distribution kit are intended to be run under BATCH. (See the *RSTS/E System User's* Guide for instructions on running the batch processor.) Since a batch control file runs detached, any output resulting from execution of the file will be directed to the batch logging file rather than to your terminal. At the completion of the BATCH run, you should look in the batch logging file for NCP status and counter information, as well as any program-generated error messages. DECnet event logging messages, however, are output to the designated logging terminal.

If you are running a RSTS/E configuration that does not have the batch processor installed, you must manually type in each of the commands, using the control file from the distribution kit as a model. In this case, output will be directed to your terminal.

When analyzing a batch logging file, you should handle any errors and unexpected results sequentially. That is, the first error occurring in a verification procedure should be corrected and the control file should be rerun from the beginning. If many errors result, it may be easier to type in each command, using the control file as a model, and correct each problem as it occurs.

1.2 Configuration Questions

Only nine RSTS/E configuration questions pertain to DECnet/E system generation. These questions are shown below in long form with the default answer displays where appropriate.

The first two questions simply ask whether you want DECnet/E support and, if so, what medium your distribution kit is on.

```
DECnet/E is a set of modules that implement the network services
protocol and user interfaces required for DECnet communication.
The DECnet/E software is sold and distributed as a separate
package. It is not included in the standard RSTS/E kit. Do you
have a DECnet/E kit and wish to include DECnet support in this
system (YES or NO)?

DECnet network support ?        #Y #


The DECnet/E package is distributed on magtape; RK05 (DK), RL01
and RL02 (DL), and RK06 and RK07 (DM) cartridge disks; or may
already have been transferred to the system disk (SY). For mag-
tape, a distinction must be made between the TU10, TE10, and
TS03 drives, that use the device name "MT"; the TU16, TE16,
TU45, and TU77 drives, that use the device name "MM"; and the
TS11 drive, that uses the device name "MS". Enter the type of
distribution medium for this DECnet/E generation (MT, MM, MS,
DK, DL, DM, or SY).

DECnet/E distribution medium ?  #MM#
```

The next three questions relate to patching the DECnet/E support within the RSTS/E operating system. These questions should be answered appropriately if an update to the RSTS/E operating system requires patching the DECnet/E software support. In the example below, "Y" was typed so that the other questions would appear. Default answers are shown for the remaining two questions.

```
The system generation process can automatically patch the gener-
ated DECnet/E support. You must have a DECnet/E patch file on
any RSTS/E file structured medium. Would you like to automati-
cally patch the DECnet/E support (YES or NO)?

DECnet/E patching ?             #??#    YES

The DECnet/E patch file can exist on the current system disk
(SY); another disk (DF, DK, DL, DM, DP, DR, or DB); TU10, TE10,
or TS03 magtape (MT); TU16, TE16, TU45, or TU77 magtape (MM); or
TS11 magtape (MS). Enter the name of the patch file medium that
has your DECnet/E patch file.

Patch file medium ?             #Identical to monitor#


Enter the file name of the DECnet/E patch file. An account num-
ber (e.g., [10,23]) or $ (for [1,2]) can be entered along with
the file name.

Patch file name ?               #$DECNTC.CMD#
```

The last four questions pertain to the configuration of your system's communications hardware. When answering these questions, consider not only your current needs and configuration but also your needs in the near future. Specifying numbers larger than you currently have has no major effect other than a slightly larger operating system but could save you the effort of doing another SYSGEN at a later date. Thus, you should answer the fourth question with the number of devices you have, or intend to have in the near future, even if you do not expect to turn them all on at once.

Note that SYSGEN checks the number of DMC11s/DMR11s and DMP11s/DMV11s currently attached to the system for the default answers for the first two questions. Note also that the third question, concerning controller type, was answered with a "DMP" response. If the question had been answered with a "DMV" response, the answer to the fourth question would have defaulted to the value 12 rather than 32.

```
The DMC11 and DMR11 network links are synchronous communication
line interfaces that implement the DDCMP line protocol in hard-
ware. RSTS/E supports up to a maximum of 16 DMC11's or DMR11's
per system. How many DMC11's or DMR11's are attached to this
system (0 to 16)?

DMC11's/DMR11's ?                #06#

The DMV11 and DMP11 network links are synchronous communication
line interfaces that implement the DDCMP line protocol in hard-
ware. In the multidrop mode, the DMV11 can support up to 12
tributaries, and the DMP11 can support up to 32 tributaries.
RSTS/E supports up to a maximum of 16 DMV11's or DMP11's per
system. How many DMV11's or DMP11's are attached to this system
(0 to 16)?

DMV11's/DMP11's ?                #00#          2


The DMP11 is the Multi-drop DDCMP interface for use on the PDP11
Unibus; the DMV11 is its Q-bus (PDP-11/23) equivalent. For auto-
configuration purposes, you must specify whether you are using
the Q-bus DMV11 controller or the Unibus DMP11 controller. In
the multidrop mode, the Q-bus DMV11 can support up to 12 tribu-
taries, and the Unibus DMP11 can support up to 32 tributaries.
Which controller type (DMV or DMP) are you using?

Controller type (DMV11/DMP11) ?#DMP#


If this DMP11 will be operated in "multipoint control" mode,
specify the maximum number of tributaries that will be "on" at
any one time, now or in the foreseeable future. The maximum num-
ber of simultaneously active tributaries is 16. More tributaries
than specified can be attached to the line, but only the number
specified can be simultaneously active. If the DMP11 will be
operated only in "multipoint slave" or in "point to point" mode,
specify 1.

DMP11 unit 00 tributaries ?      #32#

DMP11 unit 01 tributaries ?      #32#
```

The batch stream that executes after all SYSGEN dialog questions are answered will instruct you to mount the DECnet/E distribution medium on a device unit. It may be necessary to dismount the RSTS/E distribution medium in order to mount the DECnet/E tape or disk. Later in the batch stream, you would remount the RSTS/E medium.

Following the system generation procedure, you should use the RSTS/E system initialization program (INIT.SYS) to tailor the system to your operation. (See the *RSTS/E System Generation Manual* for details on running INIT.SYS.) The following DECnet/E requirements should be taken into consideration:

- Full routine nodes require space from the extended buffer pool (XBUF) for the routing matrix and other routing data structures. This is independent of other XBUF usage and must be added to other requirements to determine the total XBUF size. The amount of additional space needed is determined by the node's maximum number of circuits and the maximum node address for the network:

  ```
  words of XBUF = (maximum circuits) * (maximum node address)
                  + maximum circuits
                  + 2 * (maximum node address)
                  + 32
  ```

- Like the RSTS/E spooler programs, the DECnet/E library programs require a job area larger than 8K. Always set SWAP MAX to at least 28K.

- Use the SET PRIV option to disable nonprivilaged access to any DMP and DMV controllers. If you do not want your DMC and DMR controllers available for use by non-DECnet jobs, use the SET PRIV option to restrict access to them also.

1.3 Building the DECnet/E Library

Before building the DECnet/E library, you *must* build the following RSTS/E modules: RSX and RMS. (See the *RSTS/E System Generation Manual* for the details of this procedure.) Certain RSX utilities (such as TKB) are required to build the DECnet utilities; NCP and NML must run under the RSX Run-Time System; and RMS is required by NFT and FAL.

After these modules are built, be sure the DECnet/E distribution medium is mounted. It may be necessary to dismount the RSTS/E library medium used in the previous build and replace it with the DECnet/E distribution medium.

When the distribution medium is mounted, run the RSTS/E BUILD program and answer the appropriate questions.

- The source input device is your DECnet/E Distribution device.

- The target library account is ordinarily user account [1,2] on device SY0: — the RSTS/E system library. Task images of the network programs are installed in this account. It is possible, of course, to designate an account other that SY0:[1,2] as the DECnet library. However, if you do so, you must edit the configuration and verification control files and the DECnet startup command file to access the proper account. The existing commands to install and run the network programs will no longer work since they use the "$" account designator, specifying the RSTS/E system library (e.g., "RUN $NFT").
- The control file is DECNET.CTL from account [1,2] on the distribution device. (Note that when this file is copied from the distribution device to the target system device, it is renamed to BLD01.TMP.)
- The function is either BUILD or BUILD/PATCH, depending on whether or not there are DECnet/E patches. (Patches to DECnet/E are included in the quarterly RSTS/E autopatch distribution. Before executing the build procedures here, you should copy the DECnet/E patch files from the patch kit distribution device to SY:[200,200]. See the RSTS/E documentation for details on how to do this.)
- The question concerning NFT and FAL should be answered "RLB" if RMS has been installed as a resident library; otherwise, it should be answered "OVL".

Answer all other questions with a linefeed (LF).

The following example shows a typical DECnet/E build sequence, including patches. MM0: is the distribution device, and NFT and FAL are to be built as overlays.

```
RUN $BUILD
BUILD V7.1-07 RSTS T7.1-07
System Build <No> ? (LF)
Source Input Device <SY:> ? MM0:
Library Output Device <SY:> ? (LF)
Target System Device <SY0:> ? (LF)
Library Account <[1,2]> ? (LF)
Control File is ? MM0:[1,2] DECNET.CTL

*** Copying file MM0:[1,2] DECNET.CTL to BLD01.TMP ***

Function (Build/Patch, Patch, Build) <Build/Patch> ? BUILD/PATCH
Patch File Input Location <SY:[200,200]> ? (LF)
Run-Time System <[RSX]> ? (LF)
Do you want NFT and FAL built to use:
  RSX Run-Time System and the RMS Resident Library 'RLB'
  RSX Run-Time System with disk overlays 'OVL'
Build NFT/FAL using <OVL> ? (LF)
Additional Control File is <None> ? (LF)

*** Processing Started on 31-SEP-81 at 13:47 PM ***
```

NOTE

> You can execute the procedure described here using the DAPBLD.CTL control file from the DECnet/E distribution medium to rebuild just the file transfer software. This can be done if a new release of RMS is installed.

Chapter 2
Network Configuration

Before starting DECnet/E for the first time, you must set up the permanent parameter file ($NETPRM.SYS) to reflect the configuration of your network. This configuration process consists of the following steps:

1. Constructing the configuration control file containing the necessary NCP commands (see Section 2.1).
2. Running the control file to define values in $NETPRM.SYS (see Section 2.3).
3. Listing the network parameters to verify that the configuration was done correctly (see Section 2.3).

After the permanent parameter file is configured, you must run through the installation verification procedures (described in Chapters 3 and 4) to ensure that the communications hardware is functional and that all supported adjacent nodes are reachable. Once these procedures are complete, you can tailor the permanent parameter file to facilitate daily operation and include the network startup command file in the RSTS/E initialization procedure (see Section 2.4).

2.1 Constructing the Network Configuration Control File

The manner in which the configuration control file is constructed depends on whether or not you are currently using DECnet/E V1.1 to operate your node.

If you *have not* been using DECnet/E V1.1, print out the control file (NCPDEF.CTL) from the DECnet/E distribution kit. (Also see Appendix A.1.) This is a model configuration control file containing prototype NCP commands defining a general network environment. Edit this file to include information about your own system. For instance, you will want to use your own system's node name and number as the EXECUTOR node name and number. Comments within the model file and the information found in Section 2.2, "Node Generation Guidelines," will help you determine what modifications need to be made.

If you *have* already been using DECnet/E V1.1, the procedure for constructing the configuration control file is as follows:

1. Copy the NCU.SYS file from your V1.1 system.
2. Run the DECnet/E V2.0 conversion utility, NCUCVT, using your old NCU.SYS file as input. This utility generates an output file containing the NCP commands that will recreate your Version 1.1 configuration on the Version 2.0 system.

Since the model configuration control file supplied in the distribution kit is named NCPDEF.CTL, for the remainder of this discussion it is assumed that the name of the NCUCVT output file is also NCPDEF.CTL. The file can be named anything you wish, but if you do use the name NCPDEF.CTL, the distribution copy will be overwritten. If you wish to retain the distribution copy, you must either rename it with PIP (the Peripheral Interchange Program) before beginning NCUCVT execution or supply a different name for the NCUCVT output file. (See the *RSTS/E System User's Guide* for instructions on using PIP.)

3. Print NCPDEF.CTL and read it. It also contains commands defining certain Phase III parameters that are not applicable to your Version 1.1 system. If you wish, edit this file to change Version 2.0 defaults or to modify your existing Version 1.1 configuration. If you have DMP or DMV devices, add definitions for those lines, as described in the *DECnet/E System Manager's Guide*.

NCPDEF.CTL defines an unprivileged default account — [100,100] — for use by network programs such as MIRROR, FAL, and NML. It also defines a logging file — SY:[1,100]EVTLOG.LOG — for use by the event logger. You can use these user accounts or specify others by editing the control file. Either way, be sure that the accounts specified exist. If they do not, create them using the RSTS/E REACT utility. (See the *RSTS/E System Manager's Guide* for detail on using REACT.)

Note that logging to the console (default KB0:) is enabled by NCPDEF.CTL to permit you to see events as they happen during network installation, and file logging is enabled to provide a record for your software specialist in case of problems. Under normal circumstances, you may not wish to see this level of logging detail but it is necessary during installation verification procedures. Do *not* edit it out of the configuration control file until installation verification is complete.

2.2 Node Generation Guidelines

Certain guidelines should be followed when configuring a DECnet/E node in a network. Aside from observing basic minimum and maximum values and ensuring that related parameter values are consistent with one another (see Section 3.1), you must choose certain network parameters carefully to avoid network routing problems.

The following subsections discuss the guidelines that should be followed when specifying network parameter values with the SET EXECUTOR command. (See the *DIGITAL Network Architecture, Transport Functional Specification* for a general discussion of the Phase III routing algorithm and the details of how each network parameter affecting routing is used.)

2.2.1 Specifying Maximum Circuits

The executor parameter MAXIMUM CIRCUITS serves two purposes:

1. It controls whether your DECnet/E system will initialize itself (at start-up) as an end (nonrouting) node or as a routing node.

2. It controls the maximum number of circuits that can be turned on *simultaneously*.

A Phase III end node, by definition, has only one circuit, connecting it to one adjacent node in the network. Only network traffic to or from that node's own users and software occurs over the circuit.

A Phase III routing node, on the other hand, has more than one circuit, connecting it to multiple adjacent nodes in the network. Aside from receiving and transmitting its own network traffic, a routing node forwards data packets from other nodes. Because of this service function, a routing node must maintain a list of all nodes in the network, along with the identity of the physical line by which they can be reached. This routing table is maintained in the extended buffer pool (XBUF). Thus, DECnet/E support of a routing node uses more system resources — both memory space and processing time — than an end node. (For further information, see the *DECnet/E System Manager's Guide*.)

If your system is small or heavily used, you should consider configuring it as an end node. In a full Phase III network, you would still have access to all other nodes in the network as long as the adjacent node to which your single circuit is connected is a routing node and can perform data forwarding services for you. (Note that two connected end nodes will comprise an isolated two-node network.)

Setting the executor parameter MAXIMUM CIRCUITS equal to one causes your node to initialize itself as an end node; setting MAXIMUM CIRCUITS to a value greater than one causes the node to initialize itself as a routing node. Since this parameter defines the dynamic configuration of the DECnet/E software, it cannot be changed once the network is started. It can, however, be changed *between* network startups without repeating the SYSGEN process. Thus, your system could be a routing node one day and an end node the next.

The MAXIMUM CIRCUITS parameter also controls the number of circuits that can be turned on at one time. You can have more physical lines (or tributaries in a multipoint environment) on your system than the MAXIMUM CIRCUITS parameter specifies, but only that number can be turned on simultaneously.

2.2.2 Specifying Maximum Hops

The executor parameter MAXIMUM HOPS controls the length of the routing path that a data packet can take to its destination.

If you draw a map of your entire network, showing all physical lines connecting the nodes, it is possible to count the number of *hops* it takes to get a data packet from one node to any other node in the network. (A *hop* is the

path between two adjacent nodes — along a single circuit — no matter how long or how fast.) A *routing path* is a series of hops from a data packet's source to its destination; the length of the path is the number of hops. The *network diameter* is the length of the longest routing path. In general, the MAXIMUM HOPS parameter should have a value no less than this network diameter.

The routing software calculates the path length from the local node to every other node in the network. If the path length to any remote node is greater than MAXIMUM HOPS, that remote node is considered unreachable and the local node will be unable to communicate with it or forward data to it.

2.2.3 Specifying Maximum Cost

The executor parameter MAXIMUM COST controls the total cost of a routing path.

Each circuit on a node has a certain *cost* associated with it at that node. Cost is an arbitrary number, assigned in such a way that the routing software will prefer certain paths to others. The routing software chooses a path with minimum cost. The total cost of a routing path is the sum of the costs of each circuit on the path, as assigned at the node on the transmitting end of the circuit.

The MAXIMUM COST parameter is like the MAXIMUM HOPS parameter; if the cost of a routing path from the local node to another node in the network is higher than the MAXIMUM COST value, the remote node is considered unreachable. Thus, each node should have a MAXIMUM COST value no less than the maximum path cost of the entire network.

2.2.4 Specifying Maximum Visits

The executor parameter MAXIMUM VISITS controls the number of nodes through which a data packet can travel from its source to its destination.

Included in the header portion of each data packet is a counter of the number of nodes through which the packet has traveled. The Phase III routing software increments this counter each time the packet is forwarded to another node. If the counter exceeds the MAXIMUM VISITS parameter at a forwarding node, the data packet is discarded and a logging message is output:

```
Event type 4.0, Aged packet loss
Occurred 03-AUG-81 16:09:32.2 on node 140 (ARK)
Packet header = 4 135 140 A
```

If a line goes down on the minimum routing path, the software automatically chooses a new path if it can, possibly causing packets already in transit to back track through nodes they have already visited. Thus, the MAXIMUM VISITS parameter should be set high enough to permit a certain

amount of this type of routing activity, but low enough to catch packets caught in a loop. The value must be greater than MAXIMUM HOPS and should generally be no less than the network diameter — one to one and a half times the network diameter is a reasonable guideline.

2.2.5 Specifying Node Addresses

The executor parameter ADDRESS sets the local node's network address. Node addresses must be unique across the network. Assigning the same address to two different nodes will result in one of two situations:

1. If the path costs to send messages to the two nodes differ, messages are always directed to the node with the lower path cost.
2. If the path costs to send messages to the two nodes are the same, messages are always sent to only one of the nodes.

In both cases, messages can be received from both nodes but can be sent to only one.

NOTE

> Note that you need not assign node *names* that are unique across the network. To avoid unnecessary confusion, however, it is good practice to do so.

2.2.6 Specifying Maximum Node Address

The executor parameter MAXIMUM ADDRESS defines the size of your network. Node addresses are assigned in the range of 1 to MAXIMUM ADDRESS. Thus, if there are 32 nodes in your network, for example, MAXIMUM ADDRESS must be 32 (or more), even if only 20 of those nodes are ever on the network at one time.

The MAXIMUM ADDRESS parameter also determines the size of the routing table stored at the local node. To ensure that all routing tables are the same length, the same maximum node address should be specified for all nodes in a network.

When the maximum node address specified on one node is larger than that specified on an adjacent node, routing information passed between the two nodes will be truncated to the smaller address. Information concerning nodes with addresses higher than the smaller maximum will be lost and those nodes will be unreachable.

Since the size of the routing data base is determined by the maximum address specified, the node with the smaller maximum address will discard routing information it cannot store and will generate an event message:

```
Event type 4.5, Partial routing update loss
Occurred 03-AUG-81 16:09:38.2 on node 140 (ARK)
```

```
Circuit DMC-0
Packet header = 4 135 140 A
Highest address = 245
```

If a node name is assigned to a remote node whose address is higher than that specified in the routing table, that node will be unreachable.

2.2.7 Specifying Data Link Segment Size

The executor parameter BUFFER SIZE sets the maximum size of a data segment that the local node will receive over a link. The data link segment size should be the same for all nodes on a routing path. This ensures that the segment size is large enough for each path in the network.

When the segment size of an intermediate node is smaller than that of the source and destination nodes, a data packet larger than the smaller size will be lost on route to its destination. The node adjacent to the node with the smaller segment size will refuse to forward the data packet and will output an event message:

```
Event type 4.3, Oversized packet loss
Occurred 03-AUG-81 16:09:42.4 on node 140 (ARK)
Circuit DMC-0
Packet header = 4 135 140 A
```

If the node at which the error occurred is *not* the source, the source will observe only a failure to communicate and will not be aware of the nature of the problem. This can be especially confusing since control messages (such as the Connect Initiate Message) tend to be much shorter than user data messages and, therefore, have a higher likelihood of being transmitted, even in the above circumstance. Thus a program could establish a link but not necessarily use it.

2.3 Running the Configuration Control File

After you have constructed the configuration control file, either by editing the model supplied in the distribution kit or by running the NCUCVT conversion utility, you are ready to create the permanent parameter file. Do this by submitting NCPDEF.CTL to the batch processor. During execution of this control file, NCP creates the permanent parameter file ($NETPRM.SYS) and loads it with data according to the DEFINE commands contained in the file.

You can type additional DEFINE commands directly to NCP but it is good practice to maintain *all* commands in a single control file that completely describes *your* system.

When the control file has run to completion, you should examine the network parameters to ensure that the configuration was done correctly. Use the following NCP commands to print out the data now in $NETPRM.SYS:

LIST EXECUTOR CHARACTERISTICS
LIST KNOWN CIRCUITS CHARACTERISTICS
LIST KNOWN LINES CHARACTERISTICS
LIST KNOWN NODES CHARACTERISTICS
LIST KNOWN OBJECTS CHARACTERISTICS
LIST KNOWN LOGGING CHARACTERISTICS

Review the information generated to be sure it is correct. If any of the information is incorrect and you need to rerun the control file, you should first use PIP to delete $NETPRM.SYS. This forces NCP to create a new file. Otherwise, the NCP DEFINE commands will be applied to the previous version of $NETPRM.SYS.

2.4 Final Configuration

When installation verification procedures are complete, you may want to modify the permanent parameter file or the DECnet startup control file to facilitate daily operations. (The startup command file (DECNET.CMD) is copied from the distribution kit during the DECnet build sequence and contains commands for loading the permanent parameter file into the volatile parameter file and turning on the node. See Appendix A.7.)

The permanent parameter file is configured with all communication circuits in the OFF state. For day to day operations, however, you may want to have all or some of the circuits started immediately after the node starts up. Do this by editing the startup command file to include a SET CIRCUIT command, setting the state of each circuit you want started. These commands should be inserted following the SET EXECUTOR STATE ON command and have the following format:

SET CIRCUIT *circuit-id* STATE ON

The parameter *circuit-id* identifies the particular circuit. You could also use the SET KNOWN CIRCUITS command to set the state of all circuits:

SET KNOWN CIRCUITS STATE ON

These circuits will then be started after the permanent parameter file is loaded into the volatile parameter file with the SET SYSTEM command and after the SET EXECUTOR STATE ON command is issued.

Note that the maximum number of circuits that can be turned on at one time is determined by the MAXIMUM CIRCUITS parameter of the SET or DEFINE EXECUTOR command. If you attempt to turn on more than this number, only the specified maximum will be started. NCP will generate error messages for the remainder:

```
No room for new entry
Error detail = CIRCUIT
? No room for user on device
Error changing line/circuit parameter
```

For installation verification procedures, CONSOLE logging and FILE logging are enabled for all known events. In daily operations you may not want this volume or level of detail for event logging. Use the DEFINE LOGGING commands to specify the logging sinks to which you want DECnet to log events. If your network coordinator wants your system to log events to a central node, define this remote logging sink in the permanent parameter file as well.

While specifying the logging sinks, you can also set up the filters for each sink, defining the types of events each is to receive. Ordinarily all events should be logged to the logging FILE. You might, however, want the logging CONSOLE to print only selected event messages — line ON/OFF and executor ON/OFF messages, for example. (See the *DECnet/E System Manager's Guide* for a discussion of event logging, logging sinks, and logging filters.)

Once the startup command file and the permanent parameter file reflect your daily operational requirements, you should make one final verification that you have done everything correctly. Using the NETOFF utility, shut down the node. Run SYSTAT to verify that NSP and the event logger (EVTLOG) are no longer running. Check the event logger messages; when the node has finally shut down, the following logging message will be output:

```
Event type 2.0, Local node state change
Occurred 03-AUG-81 16:09:22.3 on node 140 (ARK)
Reason for state change: Operator command
Old node state = On
New node state = Off
```

After this, submit the final DECNET.CMD command file to the system startup control program (INIT.BAS) to restart the node. (See the *RSTS/E System Manager's Guide* for details on running INIT.BAS.) Use the NCP SHOW command to verify that everything is initialized as you want it. Verify that the event logger is logging messages to the proper logging sinks.

When this final configuration has been completed and verified, you may want to include the DECnet startup command file as input to INIT.BAS during RSTS/E startup. In this way, DECnet/E will be started automatically whenever the operating system is initialized. To do this, edit the RSTS/E startup command file, START.CTL, to include the following command as one of the final steps in system initialization:

```
@DECNET.CMD
```

Chapter 3
Local Node Operation Verification

When the permanent parameter file has been generated and the parameters have been verified to be correct, you must verify the following:

- The DECnet/E software at the local node is functioning.
- The local communications hardware is operational.
- All supported adjacent nodes are reachable.

This chapter describes the procedure for verifying the local node. Chapter 4 describes adjacent node testing.

NOTE

> During the verification procedures described in this chapter and in the following chapter, it is assumed that there is no other network activity at the local node. Thus, all event logging messages and counter activity can be attributed to the particular test being run. If, in fact, there is other network activity going on at the local node while these tests are being run, counters and event logging messages will be affected.

3.1 Starting DECnet/E on the Local Node

Before running your node in the network for the first time, start up the RSTS/E system, without DECnet/E, to ensure that it is operating correctly. Use the HARDWARE LIST option of the system initialization program (INIT.SYS) to make sure that the necessary communications devices are attached. (See the *RSTS/E System Generation Manual* for details on running INIT.SYS.) Next, start up DECnet/E by submitting the startup command file (DECNET.CMD) to the system startup control program (INIT.BAS). (See the *RSTS/E System Manager's Guide* for details on running INIT.BAS.) DECNET.CMD contains NCP commands for initializing and turning on the node.

Initializing the node is done with the SET SYSTEM command. This NCP command does two basic things: It sets up the volatile parameter file, and if the logging state has been defined as ON, it starts the event logger.

To set up the volatile parameter file, NCP first creates an empty file, using either the file specification given with the DEFINE EXECUTOR VOLATILE PARAMETER FILE command or a default file specification of SY:[0,1]NSP0.SYS. (You can determine the file specification used by issuing

the LIST EXECUTOR CHARACTERISTICS command.) After this file is created, data from the permanent parameter file ($NETPRM.SYS) is copied into it. The volatile parameter file is then installed as a RSTS/E system file.

If NCP encounters any problems in this sequence, it outputs the following message:

```
?Can't SET SYSTEM -
```

Another error message is appended to this message giving additional information about the problem. If a problem occurs, the most likely message is "System already SET" indicating that the volatile parameter file already exists. If this happens, issue the CLEAR SYSTEM command to delete the existing file, and resubmit DECNET.CMD to INIT.BAS.

Other error messages at this point are generally standard RSTS/E file operation error messages and are usually self-explanatory.

The volatile parameter file is installed as a swap file. At the completion of the batch run, you should use the UTILTY program to verify that this is the case. The LIST SWAPFILES command should display the name of the volatile parameter file in the following format:

DECnet/E file: *file-specification*

(See the *RSTS/E System Manager's Guide* for details on running UTILTY.) If you do not see this display, file a Software Performance Report (SPR) or contact your software specialist.

After the volatile parameter file is created and installed, the SET SYSTEM command starts the event logger if the logging state has been defined ON. If NCP is unable to do this, it outputs the following message:

```
?Can't start event logger
```

This message is followed by a RSTS/E standard error message giving additional information on the problem. If a problem occurs, this message is usually "?File not found." If so, make sure that the event logger task file (EVTLOG.TSK) is located on the same device and in the same account as the NCP task file (NCP.TSK). If it is not, use PIP to transfer the file to the proper device and account.

After the node is initialized, it is started with the SET EXECUTOR STATE ON command. This command first does consistency checking of the parameters in the volatile parameter file. If a parameter value is found to be out of range or inconsistent with another related parameter, NCP outputs an error message followed by an error detail message containing the name of the invalid parameter. If this error occurs, correct the DEFINE command for the invalid parameter in the configuration control file. (Table 3–1 lists the valid ranges of executor, line, and circuit parameters.)

When the SET EXECUTOR STATE ON command completes successfully, an event logging message is output:

```
Event type 2.0, Local node state change
Occurred 03-AUG-81 15:37:42.5 on node 140 (ARK)
Reason for state change: Operator command
Old node state = Off
New node state = On
```

If this message is not output, run SYSTAT to determine whether the event logger is running as a message receiver. (See the *RSTS/E System User's Guide* for information on SYSTAT.) If it is not, the event logger was not successfully spawned as a job during the SET SYSTEM sequence. (The RSTS/E Spawn Job function does not return any status to the calling job, and thus, NCP has no way of knowing whether or not the event logger was, in fact, started.) Issue the SET ALL LOGGING STATE ON command and recheck the event logger status with SYSTAT. If the event logger is still not running, contact your software specialist.

Table 3–1: Executor, Line, and Circuit Parameter Constraints

PARAMETER NAME	RESTRICTIONS
Executor Parameters	
Incoming Timer	Greater than 0
Outgoing Timer	Greater than 0
Maximum Links	Greater than 0
Delay Factor	Greater than 0
Delay Weight	Greater than 0
Inactivity Timer	Greater than 0
Retransmit Factor	Greater than 0
Routing Timer	Greater than 0
Maximum Address	Greater than or equal to the EXECUTOR address
Maximum Circuits	Greater than 0; less than or equal to 16
Maximum Cost	Greater than MAXIMUM HOPS; less than 255
Maximum Hops	Greater than 0
Maximum Visits	Greater than MAXIMUM HOPS
Maximum Buffers	Greater than or equal to 3
Buffer Size	Greater than or equal to 246; less than or equal to 568; same on all nodes throughout the network
Date Xmit Queue Max	Greater than or equal to 8
Int/LS Queue Max	Greater than or equal to 8

NOTE

The size of the routing matrix maintained by the Transport module is determined by MAXIMUM CIRCUITS and MAXIMUM ADDRESS: Routing matrix = (MAX CIRC)*(MAX ADDR). This value must be less than or equal to 4060.

PARAMETER NAME	RESTRICTIONS
Line Parameters	
Receive Buffers	Greater than or equal to 3; less than or equal to 32

(continued on next page)

Table 3–1 (Cont.): Executor, Line, and Circuit Parameter Constraints

PARAMETER NAME	RESTRICTIONS
Circuit Parameters	
Cost	Greater than 0; less than or equal to 25
Hello Timer	Greater than or equal to 1; less than or equal to 65535
Tributary	Greater than 0 for multipoint circuits

3.2 Local Node Software Verification

The DECnet/E distribution kit includes a batch control file (NTESTE.CTL) for testing the local node software (see Appendix A.2). This file contains NCP commands to do the following:

- Perform internal loopback tests that verify local data transfers through the Transport layer.
- Show local node counters, both before and after the loopback tests.
- Run NFT to verify local data transfers and job spawning capabilities.

No communication lines need to be turned on to run these tests.

Before using NTESTE.CTL, edit it as specified in the file comments to insert the name of your local node and to supply the necessary password information. When this is done, submit the control file to the batch processor.

The first part of the batch control file runs two separate local logical link loopback tests. The first test uses the LOOP EXECUTOR command to create a logical link between the local NCP program and the local MIRROR program. The second test uses the TELL prefix to execute the LOOP NODE command, specifying the local node name as the destination node. This creates *two* logical links — one between the local NCP program and the local Network Management Listener (NML) and another between NML and the local MIRROR program.

NOTE

> These commands assume that a default account has been defined for the MIRROR program at the local node with the SET/DEFINE EXECUTOR DEFAULT ACCOUNT command. If this is not the case, edit the control file to supply accounting information with the LOOP command.

If any of these logical links cannot be created, or if a link fails after the connection sequence has completed, NCP will output an error message indicating the program to which the attempted link failed (either the Listener (NML) or the MIRROR program). This message is followed by an error detail message giving further information. Table 3–2 lists the various error detail messages for these connect failures and the action that you should take if the error occurs.

Table 3–2: Local MIRROR Connect Failures

Value	Standard Text	Action
0	No node name set	File an SPR.
1	Invalid node name format	File an SPR.
2	Unrecognized node name	File an SPR.
3	Node unreachable	File an SPR.
4	Network resources	Verify that various network parameters (e.g., maximum links, transmit queue sizes) have been adequately defined.
5	Rejected by object	File an SPR.
6	Invalid object name format	File an SPR.
7	Unrecognized object	Verify that NML = object 19 MIRROR = object 25
8	Access control rejected	Verify that access control information has been specified correctly in the control file or with the SET/DEFINE EXECUTOR DEFAULT ACCOUNT command; verify that the account exists.
9	Object too busy	File an SPR.
10	No response from object	File an SPR.
11	Remote node shut down	File an SPR.
12	Node or object failed	File an SPR.
13	Disconnect by object	File an SPR.
14	Abort by object	File an SPR.
15	Abort by Management	Someone aborted the link with the ABORT LINK command; restart the test.
16	Local node shut down	Verify the node state with the SHOW EXECUTOR STATE command; restart the node.

The control file executes the SHOW EXECUTOR COUNTERS command before and after each loopback test is run. At the completion of each of the tests, you should see the following changes in the counters:

- The number of messages received and sent should increase by 2 times the value of the COUNT parameter in the LOOP command.
- The number of bytes received and sent should increase by 2 times the value of the COUNT parameter, multiplied by the length of the loopback message. (The length of the loopback message is determined by the LENGTH parameter in the LOOP command. If none is specified, a default value of 128 bytes is used.)
- The number of connect requests received and sent should increase by either one or two, depending on whether the LOOP EXECUTOR or TELL *nodename* LOOP NODE command was executed.

If any of these counters is incorrect, or if the test itself generated an error message indicating a bad loopback response, file a Software Performance Report (SPR) or contact your software specialist.

The node counters displayed with the SHOW EXECUTOR COUNTERS command also include certain error counters (e.g., "response timeouts" or "resource errors"). A nonzero value for any of these counters generally

indicates that various network parameters, such as the queue sizes or the timeout delay weight and delay factor, have not been defined adequately. Use the LIST EXECUTOR CHARACTERISTICS command to display the current network parameters, and then verify that the values are appropriate. Refer to the Appendix B of the *DECnet/E System Manager's Guide* and to Section 2.2 of this manual for guidelines to follow when defining your local node parameters.

The length of time that it takes to run these loopback tests depends upon the values of the COUNT and LENGTH parameters supplied with the LOOP command. An absolute time frame cannot be supplied. However, if it seems to you that the tests are taking an unusual amount of time, it is possible to monitor the test activity from another terminal.

- Type the NCP SHOW ACTIVE LINKS command. You should see two or four active links, depending on whether the LOOP EXECUTOR or the TELL *nodename* LOOP NODE command is being executed. (Each link is listed twice, once for each program on the link.)
- Run SYSTAT. The MIRROR program should be running and should be a declared receiver.
- Type the NCP SHOW EXECUTOR COUNTERS command two or three times. You should see the count of messages and bytes incrementing.

The final part of the local software verification tests consists of running the Network File Transfer program (NFT) to copy a file within the local node. This verifies that NFT and the File Access Listener (FAL) have been installed properly and that FAL can be started automatically as the result of an incoming request for a logical link connection.

If the file transfer software has been built and installed properly, it is located in the DECnet library, as specified during the build procedure (see Section 1.3). This is usually the RSTS/E system library — account [1,2] on the system disk — but can be any account the system manager specifies. Remember, however, that specifying a user account other than [1,2] will require changes to the distribution control files that install and run the network programs; the existing control files use the "$" account designator (specifying the system library).

If the system monitor is unable to locate the file transfer software, the following RSTS-standard error message will be generated when the batch control file attempts to run NFT:

```
?Can't find file or account
```

If this error message is output, use PIP to verify that the NFT task image file is, in fact, located in the same account being accessed by the batch control file. (See the *RSTS/E System User's Guide* for information on using PIP.) Also use the NCP SHOW KNOWN OBJECTS command to verify that FAL has been defined for automatic startup as object 17. If everything seems to be in order, but the file transfer software still cannot be run, file an SPR or contact your software specialist.

After the programs are started and the logical link is established, the batch control file (NTESTE.CTL) is copied to a temporary file in another account on the local node. From there, it is output to the batch logging file (or to the console if you are not using BATCH). The temporary file is then deleted. If the file is not output as expected and any failure conditions reported by NFT are not easily explained and corrected, file an SPR or contact your software specialist. (See the *DECnet/E Guide to User Utilities* for a list of NFT error messages.)

3.3 Local Node Hardware Verification

The DECnet/E distribution kit includes a batch control file (NTESTH.CTL) for testing the local node hardware (see Appendix A.3). This file contains NCP commands that do the following:

- Turn on the communication devices.
- Perform hardware loopback tests.
- Show node and curcuit counters, both before and after the loopback tests.

The control file is divided into two sections, with each section executing a different type of hardware loopback testing. The first section performs a controller-level loopback test to verify that the device can be turned on and that the firmware can transfer data properly. The second section performs a hardware-level loopback test to verify that the physical line is operating. Before submitting the control file to the batch processor, you must make the hardware adjustments necessary for the hardware-level testing. (See Appendix C for the details on how to do this.) These adjustments do not interfere with the controller-level tests. You must also edit the control file to include the name of your local node and the name of the communication device to be tested. The test must be duplicated for each device on the local system.

As mentioned in Appendix C (see Section C.2.2), it may be necessary to run several hardware-level loopback tests, moving the loopback point progressively further out each time, to completely test a particular line. If so, duplicate the hardware-level loopback section of the control file, or separate the file into two pieces and run the second section as many times as required.

When the file is edited and the hardware is configured properly, submit the control file to the batch processor. At the beginning of each test, the circuit being tested should be in the OFF state.

NOTE

> If you have been running DECnet/E V1.1 and are certain that your communication hardware is working properly, you can delete the section of the control file that runs hardware-level tests. You should, however, run at least one controller-level loopback test to verify that the Transport module and the device driver software are operating correctly.

The procedure followed in each section of the control file is basically the same, and the results of each test can be examined in exactly the same fashion.

The first step in each section is to *condition* the line for testing. (See Section C.1 for details on conditioning a line and the command sequence used for each type of loopback test.) After this the circuit is turned on with the SET CIRCUIT STATE command. This command first does consistency checking on the line and circuit parameters. If any values are out of range or inconsistent with other related parameters, the SET CIRCUIT command will fail. If this happens, use the SHOW LINE CHARACTERISTICS and SHOW CIRCUIT CHARACTERISTICS commands to determine which parameter is in error. (Table 3–1 lists the valid ranges of line and circuit parameters.) Correct the parameter value with the DEFINE command and resubmit the control file.

After issuing the command to turn the circuit on, the control file executes a time delay to permit the circuit to initialize. Next, the SHOW CIRCUIT STATUS command is issued. The circuit *state* should be ON and the *adjacent node* should be the name of the local node. If the state of the circuit is still OFF, it is possible that the time delay is not long enough to permit the circuit to completely initialize. Change the delay factor in the batch control file, turn the circuit off by issuing the SET CIRCUIT STATE OFF command, and resubmit the control file. If the circuit is still OFF, it is possible that the circuit is not owned by DECnet. Issue the SET CIRCUIT OWNER EXECUTOR command, and then resubmit the control file.

If the state of the circuit is ON-STARTING, either the line is not full-duplex point-to-point, or there is a hardware problem in either the controller loopback feature or in the installation of the loopback connector (depending on the test being run). Correct the problem, if possible, and resubmit the control file. (As noted in Appendix C (see Section C.2.1), if the circuit cannot be configured for full-duplex communication to the remote system, you cannot perform certain hardware-level loopback tests.)

When the circuit has initialized and changed state, an event message is generated.

After the circuit is on and initialized, the control file defines a temporary loopback node name to the circuit. If this fails, NCP will generally output one of the following error messages:

```
?Resource error
```

or

```
?Duplicate node name
```

A resource error usually indicates that NCP could not get enough space from the RSTS/E small buffer pool to create the dynamic data structures necessary for the temporary node name. Since the number of buffers in the small buffer pool is related to overall system loading, you should simply wait a short time, reset the circuit state to OFF, and resubmit the control file.

If the error message output indicates a duplicate node name, issue the SHOW KNOWN NODES command to determine which node names and aliases are currently in use. Edit the control file so that the SET NODE CIRCUIT command that creates the temporary loopback node name specifies a name that is not being used. Reset the circuit state to OFF, and resubmit the control file.

Once the temporary loopback node name is defined, the control file issues the SHOW NODE COUNTERS and the SHOW CIRCUIT COUNTERS commands to display the current node and circuit counters. It then executes the actual loopback test. The LOOP command issued to execute the actual loopback test assumes that a default account has been defined for the MIRROR program at the local node with the SET/DEFINE EXECUTOR DEFAULT ACCOUNT command. If this is not the case, edit the control file to supply accounting information with this command.

When the test has completed, the node and circuit counters are displayed again. You should see the following changes in the counters:

Node counters

- The number of messages received and sent should increase by 2 times the value of the COUNT parameter in the LOOP command.
- The number of bytes received and sent should increase by 2 times the value of the COUNT parameter multiplied by the length of the loopback message. (The length of the loopback message is determined by the LENGTH parameter in the LOOP command. If none is specified, a default value of 128 bytes is used.)
- The number of the connect requests received and sent should increase by one.

Circuit counters

- The arriving packets received and departing sent should increase by a value equal to, or a multiple of, the COUNT parameter in the LOOP command.
- The data blocks received and sent should increase by a value equal to, or a multiple of, the COUNT parameter in the LOOP command, plus some

small additional amount (5 or 6) representing logical link overhead (Connect Initiate Messages, and so on).

- The number of bytes recieved and sent should increase by 2 times the value of the COUNT parameter, multipled by the length of the loopback message, plus some additional amount (approximately 30 times the COUNT parameter) representing logical link overhead (acknowledgments, and so on).

If any of these counters are incorrect, or if the test itself generated an error message indicating a bad loopback response, file a Software Performance Report (SPR) or contact your software specialist.

The counters displayed also include certain error counters (e.g., "response timeouts" or "congestion loss"). A nonzero value for any of these counters generally indicates that various network and circuit parameters have not been adequately defined. Use the LIST NODE CHARACTERISTICS and LIST CIRCUIT CHARACTERISTICS commands to display the current parameters. Refer to Appendix B of the *DECnet/E System Manager's Guide* and to Section 2.2 of this manual for guidelines to follow when defining local node and circuit parameters.

The final steps in the testing sequence consist of clearing the temporary loopback node name, turning the circuit off, and resetting the circuit controller mode to NORMAL. If the line being tested normally operates in half-duplex mode, you should then issue a SET LINE DUPLEX HALF command.

Chapter 4
Adjacent Node Testing

Once you have determined that the local node software and hardware are operating correctly, you must ensure that the connections to all adjacent nodes have been established and that you can transfer data between the local node and all its neighbors. Note that in this chapter we are discussing *logically* adjacent nodes, where a logically adjacent node is defined as a node to which direct connections can be established without routing through another node. For multipoint tributaries, only the control station is considered to be a logically adjacent node. Other tributaries on the same multipoint line are not considered adjacent to each other. All tributaries are adjacent to the control station, however.

Before the procedures described in this chapter can be executed, all communication lines to be used must pass the local loopback tests described in Chapter 3. Also, all adjacent nodes must be operating correctly. If you are installing DECnet on multiple nodes of your network, you should install one node at a time, completely testing the local hardware and software of each node before installing the next. If no nodes in the network are currently known to be operating DECnet correctly, you should run all the local tests specified in Chapter 3 on each of two nodes before attempting to connect them.

The DECnet/E distribution kit contains three batch control files for testing the connections to adjacent nodes:

NTESTA.CTL	For adjacent Phase III nodes (Appendix A.4)
NTEST2.CTL	For adjacent Phase II nodes running DECnet/E V1.1, DECnet-RT V1.1, and DECnet-20 V2.0 (Appendix A.5)
DTSDTR.CTL	For adjacent Phase II nodes not running one of the DECnet implementations mentioned above (Appendix A.6)

You must run one of these files for each adjacent node. Be certain to remove any loopback connectors installed during local hardware verification.

All the procedures described here are designed to be initiated and executed from the local node. However, an operator should be present at each remote node to handle problem isolation. To facilitate problem isolation for remote loop testing, ask the remote operator to enable event logging on his node if it has logging capabilities. (Phase II nodes and some Phase III nodes do not.)

Before testing the connection to an adjacent node, you must know whether the node in question is a Phase II or a Phase III DECnet node, and if it is a Phase II node, what DECnet implementation it is running. If you do not already know this information, ask either the network manager or the system manager of the adjacent node.

When you have determined the phase and implementation of each of the nodes adjacent to your local node, edit the appropriate file for each node with which you want to test connections. Do this by making the necessary number of copies of each control file or by duplicating the test sections within the file itself. Insert the name of the local node, the name of the communication device, the name of the adjacent node, and any accounting information necessary.

NOTE

The LOOP commands in the control files that execute the actual loopback tests assume that a default account has been defined for the looper program at the remote node. If this is not the case, edit these commands in the control file to supply accounting information with the LOOP command.

4.1 Tests with Remote Phase III Nodes

You test connections with adjacent Phase III nodes using batch control file NTESTA.CTL from the DECnet/E distribution kit. This file contains NCP and system level commands to do the following:

- Turn on the curcuit to the node.
- Show the status of the circuit and of the adjacent node.
- Run remote node loopback tests that test the software of both nodes.
- Show node and circuit counters, both before and after the loopback tests.
- Run NFT to copy a file to and from the adjacent node.

Two types of remote node loopback tests are run with each adjacent Phase III node. The first test creates a logical link between the local NCP and the *local* MIRROR program; the loopback data is turned around in the remote node Transport layer.

NOTE

If the adjacent node is not a routing node, the remote Transport software will not forward messages back to the local node. Thus, this first test will fail and the following message will be output:

```
MIRROR connect failed -- Node unreachable
```

The second test creates a link between the local NCP and the *remote* MIRROR program; in this case, the loopback data is turned around by the MIRROR program in the remote node User layer. Verification of both these tests follows basically the same procedure as that followed for local hardware verification (see Section 3.3).

Before submitting the control file to the batch processor, be sure that all loopback connectors installed during hardware verification have been removed and that the circuit to the remote node is in the OFF state.

The first step in the control file sequence is to turn the circuit on with the SET CIRCUIT STATE command. This command first does consistency checking on the line and circuit parameters. If the command fails because a parameter value is invalid, correct the parameter as described in Section 3.3, and resubmit the control file.

After turning the circuit on, the control file executes a time delay to permit the circuit to completely initialize. It then issues the SHOW CIRCUIT STATUS command. At this point, the circuit *state* should be ON and the *adjacent node* should be the name of the node with which you are testing. As with local node hardware verification described in Section 3.3, if the state of the circuit is still OFF, it is possible that the control file's time delay is too short. Change the delay factor in the control file, turn the circuit off by issuing the SET CIRCUIT STATE OFF command, and resubmit the control file.

If the state of the circuit is ON-STARTING, check for an event logging message indicating that the circuit has gone down or has failed to initialize. The event message contains further explanatory information. Correct the problem, turn the circuit off by issuing the SET CIRCUIT STATE OFF command, and resubmit the control file.

Once the circuit is on and initialized, the control file defines a temporary loopback node name to the circuit and issues the LOOP command. This runs a loopback test with the remote Transport layer, creating a link with the *local* MIRROR program. (If the definition of the loopback node name fails, see Section 3.3. If the link connection to the local MIRROR program fails, see Section 3.2.)

Node, circuit and executor counters are displayed, both before and after the loopback test. In addition, node counters are displayed for both the actual node and for the loopback node. The circuit counters and the loopback node counters should be verified in the same way as for local hardware verification (see Section 3.3). In this case, however, error counters for the circuit do not necessarily indicate problems with the definition of circuit parameters. They can indicate "noise" on the line (that is, electrical interference) or buffer shortages at the remote node.

Assuming there is no other network activity between the two nodes, the executor counters and the adjacent node counters should show no traffic. However, as discussed in Sections 3.2 and 3.3, various error counters may indicate that certain network parameters have been inadequately defined. Display the parameters of each node, and refer to Appendix B of the *DECnet/E System Manager's Guide* and to Section 2.2 of this manual for guidelines to follow when defining network parameters.

After the counters are displayed, the control file clears the temporary loopback node name and then issues another LOOP command. This runs the sec-

ond loopback test, creating a logical link between the local NCP and the remote MIRROR program. If this link cannot be created, or if it fails after the connection sequence has completed, NCP outputs an error message. This message is followed by an error detail message giving further information on the reason for the failure. Table 4–1 lists the various error detail messages and the action that you should take if the error occurs.

Table 4–1: Remote MIRROR Connect Failures

Value	Standard Text	Action
0	No node name set	File an SPR.
1	Invalide node name format	Correct the node name specified in the control file.
2	Unrecognized node name	Correct the node name specified in the control file.
3	Node unreachable	Verify that the node address is less than or equal to MAXIMUM ADDRESS and that circuit cost is less than or equal to MAXIMUM COST; check for event logging message, either locally or at the remote node, indicating a circuit failure.
4	Network resources	Verify that various network parameters (e.g., maximum links, transmit queue sizes) have been adequately defined.
5	Rejected by object	Check with the remote system operator for a possible explanation of the problem; if none can be determined, file an SPR.
6	Invalid object name format	File an SPR.
7	Unrecognized object	Verify that the remote MIRROR program is defined as object 25.
8	Access control rejected	Verify that access control information has been specified correctly in the control file or with the SET/DEFINE EXECUTOR DEFAULT ACCOUNT command; verify that the account exists.
9	Object too busy	Check with the remote system operator for a possible explanation of the problem; if none can be determined, file an SPR.
10	No response from object	Check with the remote system operator for a possible explanation of the problem; if none can be determined, file an SPR.
11	Remote node shut down	Verify the node state with the SHOW NODE STATUS command; have remote operator restart the node.
12	Node or object failed	Check with the remote system operator for a possible explanation of the problem; if none can be determined, file an SPR.
13	Disconnect by object	Check with the remote system operator for a possible explanation of the problem; if none can be determined, file an SPR.
14	Abort by object	Check with the remote system operator for a possible explanation of the problem; if none can be determined, file an SPR.
15	Abort by Management	Someone aborted the link with the ABORT LINK command; restart the test.
16	Local node shut down	Verify the node state with the SHOW EXECUTOR STATUS command; restart the node.

Executor, adjacent node and circuit counters are displayed both before and after the loopback test is run. You should see the following changes in the node and circuit counters:

Node counters

- The number of messages received and sent should increase by the value of the COUNT parameter in the LOOP command.
- The number of bytes received and sent should increase by the value of the COUNT parameter, multiplied by the length of the loopback message. (The length of the loopback message is determined by the LENGTH parameter in the LOOP command. If none is specified, a default value of 128 bytes is used.)
- The number of the connect requests received and sent should increase by one.

Circuit counters

- The arriving packets received and departing packets sent should increase by a value equal to, or a multiple of, the COUNT parameter in the LOOP command.
- The data blocks received and sent should increase by a value equal to, or a multiple of, the COUNT parameter in the LOOP command, plus some small additional amount (2 or 3) representing logical link overhead (Connect Initiate Messages, and so on).
- The number of bytes received and sent should increase by the value of the COUNT parameter, multipled by the length of the loopback message, plus some additional amount (approximately 15 times the COUNT parameter) representing logical link overhead (acknowledgments, and so on).

If any of these counters is incorrect, or if the test itself generated an error message indicating a bad loopback response, file a Software Performance Report (SPR) or contact your software specialist.

Again, assuming there is no other network activity between the two nodes, the executor counters should show no traffic, but various error counters may indicate that certain network parameters have been defined inadequately (see Section 3.2).

The final part of the remote Phase III verification procedure consists of running the Network File Transfer program (NFT) to verify that a file can be copied between nodes. The batch control file (NTESTA.CTL) is copied to a temporary file at the remote node. From there it is copied back to the local node and output to the batch logging file (or to the terminal console if you are not using BATCH). It is then deleted from the remote node. If the file is not output as expected and any failure conditions reported by NFT are not easily explained and corrected, file an SPR or contact your software specialist. (See the *DECnet/E Guide to User Utilities* for a list of NFT error messages.)

When all tests have been run, the control file turns the circuit off by issuing the SET CIRCUIT STATE OFF command.

4.2 Tests with Remote Phase II Nodes

You test connections with adjacent Phase II nodes using either control file NTEST2.CTL or control file DTSDTR.CTL from the DECnet/E distribution kit. NTEST2.CTL should be used to test connections with Phase II nodes running DECnet/E V1.1, DECnet-RT V1.1, and DECnet-20 V2.0. It contains NCP and system level commands to do the following:

- Turn on the circuit to the node.
- Show the status of the circuit and of the adjacent node.
- Run a Phase II remote node loopback test.
- Show node and circuit counters.
- Run NFT to copy a file to and from the adjacent node.

Phase II remote node loopback tests create a link between the local NCP and some type of remote looper program (e.g., the DECnet/E Version 1.1 NCU program). The loopback data is turned around by the remote node User layer.

The DTSDTR.CTL batch control file should be used to test connections with adjacent Phase II nodes running any supported DECnet product not listed above. It contains NCP and system level commands to do the following:

- Turn on the circuit to the node.
- Show the status of the circuit and of the adjacent node.
- Run the DTS/DTR test tool to transfer data to and from the adjacent node. (See Appendix E for a discussion of DTS/DTR.)
- Show node and circuit counters.
- Run NFT to copy a file to and from the adjacent node.

Verification of node and circuit counters resulting from these tests follows exactly the same procedure as that used for the second portion of Phase III remote node loopback testing (see Section 4.1).

NOTE

> Since Phase II nodes do not include the expanded loopback capabilities found in Phase III, the DECnet utility programs are used as logical link loopback tools. Table 4–2 lists the utilities that can be used for this purpose, in descending order of preference. The table also reflects their availability on each of the supported DECnet products.

Table 4–2: Summary of Utility Availability on Remote Nodes

	RSX DECnet	DECnet–20	DECnet–RT	DECnet–VAX	DECnet/E V1.1
DTS/DTR	Yes	Yes(2)	Yes(1)	Yes	No
NFT	Yes	Yes(3)	Yes	Yes	Yes
TLK/LSN	Yes	No	Yes	No	Yes

(1) DTS is available on Version 1.1 only.
(2) Available from software specialist.
(3) Available in DECnet–20 V2.

Appendix A
Distribution Kit Control Files

A.1 Network Configuration Control File (NCPDEF.CTL)

```
$JOB/CCL/NAME=NCPDEF/NOLIMIT/ERROR:FATAL
$!
$! Define default configuration for permanent parameter file.
$!
$RUN $NCP
!
! ********************************************************************
!
!        NCPDEF.CTL       V2.0600
!
! Copyright (C) 1981 by Digital Equipment Corporation, Maynard, Mass.
!
! Edit history:
!
! 2.0600 24-Nov-81     DECnet/E V2.0 Release            MLS
!
! ********************************************************************
!
! This file is a prototype NCP command file designed to generate
! a DECnet/E V2.0 node configuration file, ie the permanent parameter
! file $NETPRM.SYS.  The user should modify this file as indicated
! to define the node's individual configuration and the surrounding
! network configuration.  See the DECnet/E Network Installation Manual
! for further information on the use of this file.  The prototype
! configuration below is used in all prototype installation test
! command files in this manual.
!
! ********************************************************************
!
! Prototype local node assumptions:
!        ALPHA    = your node name, ie the executor node name
!        1        = the executor node number
!        DMR-0    = point-to-point device
!        DMR-1    = point-to-point device (answers phone - slow line)
!        DMP-0    = multipoint device (master)
!        DMP-1    = multipoint device (slave)
!
! Prototype network node assumptions:
!        BETA     =  2 = an adjacent Phase II node via DMR-0
!        GAMMA    =  3 = an adjacent Phase II node via DMR-1
!        DELTA    =  4 = an adjacent Phase III tributary via DMP-0
!        EPSILN   =  5 = an adjacent Phase III tributary via DMP-0
!        ZETA     =  6 = an adjacent Phase III master via DMP-1
!        ETA      =  7 = a non-adjacent Phase III node
!        THETA    =  8 = a non-adjacent Phase III node
!        IOTA     =  9 = a non-adjacent Phase III node
!        KAPPA    = 10 = a non-adjacent Phase III node
!        LAMBDA   = 11 = a non-adjacent Phase III node
!        MU       = 12 = a non-adjacent Phase III node
```

```
! ! Physical connectivity diagram:
!
! ALPHA -------------------- BETA ------------ MU
! ! ! !                      !                 !
! ! ! !                      ----------------- LAMBDA --------- KAPPA
! ! ! !                                           !               !
! ! ! ---------------------- GAMMA                !               !
! ! !                                             !               !
! ! ------------------------ DELTA                !              IOTA
! !                       !                       !
! !                       -- EPSILON              !
! !                                               !
! -------------------------- ZETA                 !
!                    !  ! !                       !
!                    !  ! !-- ETA                 !
!                    !  !                         !
!                    !  ---- THETA                !
!                    !                            !
!                    ------------------------------
!
!
! Prototype network parameters:
!       network diameter = 5     (eg DELTA-IOTA path)
!
!
! *********************************************************************
!
! DEFINE PERMANENT PARAMETER FILE command
!
!       This command creates the permanent parameter file, $NETPRM.SYS.
!
DEFINE PERMANENT PARAMETER FILE
! *********************************************************************
!
! DEFINE EXECUTOR commands
!
!       These parameters describe your node.
!
DEFINE EXECUTOR ADDRESS 1
DEFINE EXECUTOR NAME ALPHA
DEFINE EXECUTOR IDENTIFICATION "DECnet/E V2.0 Prototype"
DEFINE EXECUTOR ALIAS $LOCAL
!
!       These parameters are based on the network configuration.
!       Constraints:    All values > 0
!                       MAX CIRCUITS <= 16
!                       MAX ADDR >= EXECUTOR ADDRESS
!                       MAX VISITS > MAX HOPS
!                       MAX COST >= MAX HOPS, and < 255.
!                       (MAX ADDR) * (MAX CIRCUITS) <= 4060.
!
DEFINE EXECUTOR MAXIMUM CIRCUITS 5
DEFINE EXECUTOR MAXIMUM ADDRESS 12
DEFINE EXECUTOR MAXIMUM HOPS 6
DEFINE EXECUTOR MAXIMUM VISITS 8
DEFINE EXECUTOR MAXIMUM COST 20
!
```

```
!        These parameters control local system resource usage.
!        The buffer size should be the same across the network.
!        Constraints:    All values > 0  (except default account not required)
!                        BUFFER SIZE >= 246., and <= 568.
!                        DATA TRANSMIT QUEUE MAXIMUM <= 8.
!                        INTERRUPT TRANSMIT QUEUE MAXIMUM <= 8.
!                        MAXIMUM BUFFERS >= 3
!
DEFINE EXECUTOR BUFFER SIZE 568
DEFINE EXECUTOR DEFAULT ACCOUNT [100,100]
DEFINE EXECUTOR MAXIMUM LINKS 32
DEFINE EXECUTOR MAXIMUM BUFFERS 16
DEFINE EXECUTOR DATA TRANSMIT QUEUE MAXIMUM 8
DEFINE EXECUTOR INTERRUPT TRANSMIT QUEUE MAXIMUM 5
!
!        These parameters control NSP and Transport processing.
!        Constraints:    All values > 0
!
DEFINE EXECUTOR INCOMING TIMER 120
DEFINE EXECUTOR OUTGOING TIMER 120
DEFINE EXECUTOR INACTIVITY TIMER 120
DEFINE EXECUTOR DELAY FACTOR 48
DEFINE EXECUTOR DELAY WEIGHT 3
DEFINE EXECUTOR RETRANSMIT FACTOR 5
DEFINE EXECUTOR ROUTING TIMER 120
!
! *******************************************************************
!
! DEFINE LINE commands
!
!        These parameters define your physical devices.
!        Constraint:     RECEIVE BUFFERS >= 3, and <= 32.
!
DEFINE LINE DMR-0 DUPLEX FULL
DEFINE LINE DMR-0 VERIFICATION OFF
DEFINE LINE DMR-0 RECEIVE BUFFERS 4
!
!        Note:   the node GAMMA would have its connected device set
!                to "VERIFICATION ORIGINATE".
!
DEFINE LINE DMR-1 DUPLEX HALF
DEFINE LINE DMR-1 VERIFICATION ANSWER
DEFINE LINE DMR-1 RECEIVE BUFFERS 3
!
DEFINE LINE DMP-0 PROTOCOL DDCMP CONTROL
DEFINE LINE DMP-0 DUPLEX FULL
DEFINE LINE DMP-0 VERIFICATION OFF
DEFINE LINE DMP-0 RECEIVE BUFFERS 5
!
DEFINE LINE DMP-1 PROTOCOL DDCMP TRIBUTARY
DEFINE LINE DMP-1 DUPLEX FULL
DEFINE LINE DMP-1 VERIFICATION OFF
DEFINE LINE DMP-1 RECEIVE BUFFERS 4
!
! *******************************************************************
!
DEFINE CIRCUIT commands:
!
!        These commands define your logical circuits.
!        Constraints:    COST >= 0, and <= 25.
!                        HELLO TIMER >= 1, and <= 65535.
!                        If multipoint control or tributary,
!                          then TRIBUTARY ADDRESS > 0.
```

```
!
DEFINE CIRCUIT DMR-0 OWNER EXECUTOR
DEFINE CIRCUIT DMR-0 COST 2
DEFINE CIRCUIT DMR-0 AUTORESTART ON
DEFINE CIRCUIT DMR-0 HELLO TIMER 15
!
DEFINE CIRCUIT DMR-1 OWNER EXECUTOR
DEFINE CIRCUIT DMR-1 COST 3
DEFINE CIRCUIT DMR-1 AUTORESTART ON
DEFINE CIRCUIT DMR-1 HELLO TIMER 60
!
DEFINE CIRCUIT DMP-0.0 OWNER EXECUTOR
DEFINE CIRCUIT DMP-0.0 TRIBUTARY 2
DEFINE CIRCUIT DMP-0.0 COST 1
DEFINE CIRCUIT DMP-0.0 AUTORESTART ON
DEFINE CIRCUIT DMP-0.0 HELLO TIMER 15
!
DEFINE CIRCUIT DMP-0.1 OWNER EXECUTOR
DEFINE CIRCUIT DMP-0.1 TRIBUTARY 3
DEFINE CIRCUIT DMP-0.1 COST 1
DEFINE CIRCUIT DMP-0.1 AUTORESTART ON
DEFINE CIRCUIT DMP-0.1 HELLO TIMER 15
!
DEFINE CIRCUIT DMP-1.0 OWNER EXECUTOR
DEFINE CIRCUIT DMP-1.0 TRIBUTARY 11
DEFINE CIRCUIT DMP-1.0 COST 1
DEFINE CIRCUIT DMP-1.0 AUTORESTART ON
DEFINE CIRCUIT DMP-1.0 HELLO TIMER 15
!
! **********************************************************************
!
! DEFINE OBJECT commands:
!
!        These commands define the standard DECnet/E objects.
!        They assume that the "$" account (SY:[1,2]) was chosen as
!        the DECnet/E package location during the SYSGEN BUILD procedure.
!        If a different package location was chosen, the filespecs in
!        these commands should be modified accordingly.
!
DEFINE OBJECT 16 NAME LSN
DEFINE OBJECT 16 FILE $LSN.BAC
DEFINE OBJECT 16 PARAMETERS 29000,0
!
DEFINE OBJECT 17 NAME FAL
DEFINE OBJECT 17 FILE $FAL.TSK
DEFINE OBJECT 17 PARAMETERS 0,511
!
DEFINE OBJECT 19 NAME NML
DEFINE OBJECT 19 FILE $NCP.TSK
DEFINE OBJECT 19 PARAMETERS 29000,0
!
DEFINE OBJECT 20 NAME NETCPY
DEFINE OBJECT 20 FILE $NETCPY.BAC
DEFINE OBJECT 20 PARAMETERS 29000,0
!
DEFINE OBJECT 23 NAME NPKDVR
DEFINE OBJECT 23 FILE $NPKDVR.TSK
!
DEFINE OBJECT 25 NAME MIRROR
DEFINE OBJECT 25 FILE $MIRROR.TSK
!
DEFINE OBJECT 26 NAME EVTLSN
DEFINE OBJECT 26 FILE $EVTLOG.TSK
!
```

```
DEFINE OBJECT 63 NAME DTR
DEFINE OBJECT 63 FILE $DTRECV.TSK
DEFINE OBJECT 63 PARAMETERS 29000,25600
!
! *********************************************************************
!
! DEFINE NODE commands:
!
!       These commands define the other nodes in the network.
!
DEFINE NODE  2 NAME BETA
DEFINE NODE BETA ALIAS DATBAS
!
!       Note:   If node GAMMA were a DECnet/E node:
!               its dial-in line would be VERIFICATION ORIGINATE;
!               and for its definition of node ALPHA, it would have
!               "RECEIVE ORIGINATE PASSWORD WESAYIT" and
!               "TRANSMIT ORIGINATE PASSWORD YOUSAYIT".
!
DEFINE NODE  3 NAME GAMMA
DEFINE NODE GAMMA RECEIVE ANSWER PASSWORD YOUSAYIT
DEFINE NODE GAMMA TRANSMIT ANSWER PASSWORD WESAYIT
!
DEFINE NODE  4 NAME DELTA
DEFINE NODE  5 NAME EPSILN
!
DEFINE NODE  6 NAME ZETA
DEFINE NODE  6 ALIAS MASTER
!
DEFINE NODE  7 NAME ETA
DEFINE NODE  8 NAME THETA
DEFINE NODE  9 NAME IOTA
DEFINE NODE 10 NAME KAPPA
DEFINE NODE 11 NAME LAMBDA
DEFINE NODE 12 NAME MU
!
! *********************************************************************
!
! DEFINE LOGGING commands:
!
!       These commands control event logger processing.
!
!       This command causes the event logger to be automatically
!       started by NCP when the "SET SYSTEM" command is executed.
!
DEFINE ALL LOGGING STATE ON
!
!       These commands define console and file logging sinks and
!       enable active logging to them.
!
DEFINE LOGGING CONSOLE NAME KB0:
DEFINE LOGGING CONSOLE STATE ON
DEFINE LOGGING CONSOLE KNOWN EVENTS
!
DEFINE LOGGING FILE NAME SY:[1,100]EVTLOG.LOG
DEFINE LOGGING FILE STATE ON
DEFINE LOGGING FILE KNOWN EVENTS
!
! *********************************************************************
!
$EOD
$!
$EOJ
```

A.2 Local Software Verification Control File (NTESTE.CTL)

```
$JOB/CCL/NAME=NTESTE/NOLIMIT/ERROR:FATAL
$!
$! ******************************************************************
$!
$!       NTESTE.CTL      V2.000
$!
$! Copyright (C) 1981 by Digital Equipment Corporation, Maynard, Mass.
$!
$! Edit history:
$!
$! 2.000   24-Nov-81       DECnet/E V2.0 Release             MLS
$!
$! ******************************************************************
$!
$! This file is a prototype BATCH control file designed to verify that
$! the DECnet/E software on the local node has been installed correctly.
$! The command procedure assumes that the network has been started.
$!
$! The user should modify this file as indicated in the comments
$! to define the node's individual configuration and the surrounding
$! network configuration.  See the DECnet/E Network Installation Manual
$! for further information on the use of this file.  The prototype
$! configuration below is used in all prototype installation test
$! command files in this manual.
$!
$! ******************************************************************
$!
$! Prototype local node assumptions:
$!       ALPHA    = your node name, ie the executor node name
$!       1        = the executor node number
$!
$! ******************************************************************
$!
$! This section contains NCP commands to perform internal loopback tests,
$! without using any communication lines.  This will verify that the
$! network management tasks are installed where they are expected, that
$! network tasks can be spawned by NSP, and that NSP can create logical
$! links and transfer data over them. The "SHOW COUNTERS" commands display
$! the number of messages transferred; this should be compared with the
$! "COUNT" parameter in the "LOOP" command.
$!
$RUN $NCP
!
! The LOOP command runs a loopback test between the local NCP and MIRROR
! tasks.  Accounting information to be passed to the MIRROR task has been
! omitted.  If a default account has not been defined for network managment
! on your node, you should insert accounting information in this command.
!
SHOW EXECUTOR COUNTERS
LOOP EXECUTOR COUNT 20
SHOW EXECUTOR COUNTERS
!
! This LOOP command creates a logical link between the local NCP and NML
! tasks, and runs a loopback test between the local NML and MIRROR tasks.
! Modify the SET command to insert a privileged user-id and password, and
! to insert the name of your local node in place of "ALPHA". This user-id
! and password is passed to the NML program, which requires a PRIVILEGED
! account to run loopback tests.  A second set of accounting information
! may be used in the LOOP command line, as in the example above.  That
! accounting information would be passed to the MIRROR task, and may be
! omitted if a default account has been defined.
```

```
!
SET EXECUTOR NODE ALPHA USER [1,100] PASSWORD DNETST
SHOW EXECUTOR COUNTERS
LOOP EXECUTOR COUNT 20
SHOW EXECUTOR COUNTERS
CLEAR EXECUTOR NODE
!
$EOD
$!
$! **********************************************************************
$!
$! This section contains NFT commands to verify that the network file
$! access routines are installed where they are expected, and that a
$! file can be copied within the local node.  Modify the commands to
$! insert a valid user-id and password, to insert the name of your
$! local node in place of "ALPHA", and to insure that a valid file
$! specification appears as the source for the copy operation.
$!
$! These commands create a logical link between the local NFT and FAL
$! tasks.
$!
$RUN $NFT
COPY ALPHA::NETTST.TMP=[1,2]NTESTE.CTL
[1,100]
DNETST

TYPE ALPHA::NETTST.TMP
DELETE ALPHA::NETTST.TMP
$EOD
$!
$! **********************************************************************
$!
$EOJ
```

A.3 Local Hardware Verification Control File (NTESTH.CTL)

```
$JOB/CCL/NAME=NTESTH/NOLIMIT/ERROR:FATAL
$!
$! **********************************************************************
$!
$!        NTESTH.CTL        V2.000
$!

$! Copyright (C) 1981 by Digital Equipment Corporation, Maynard, Mass.
$!
$! Edit history:
$!
$! 2.000   24-Nov-81        DECnet/E V2.0 Release              MLS
$!
$! **********************************************************************
$!
$! This file is a prototype BATCH control file designed to verify that the
$! communication line hardware/firmware on a DECnet/E V2.0 node can be
$! turned on, and will transfer data in local loopback tests.
$! The user should modify this file as indicated in the comments
$! to define the node's individual configuration and the surrounding
$! network configuration.  See the DECnet/E Network Installation Manual
$! for further information on the use of this file.  The prototype
$! configuration below is used in all prototype installation test
$! command files in this manual.
$!
$! **********************************************************************
```

```
$!
$! Prototype local node assumptions:
$!       ALPHA    = your node name, ie the executor node name
$!       1        = the executor node number
$!       DMR-0    = point-to-point device
$!       DMR-1    = point-to-point device (answers phone - slow line)
$!       DMP-0    = multipoint device (master)
$!       DMP-1    = multipoint device (slave)
$!
$! The tests below are defined for the DMR-0 device only.  They should
$! be repeated as necessary for other devices on the system.
$! Note that the second test below requires the MANUAL installation
$! of a hardware turnaround connector, or the MANUAL setting of a
$! switch on a local modem.  This should be done before running
$! this command file; it will not interfere with the operation of
$! the first test, where data is turned around within the device.
$!
$! ******************************************************************
$!
$! This section performs a controller loopback test, which verifies
$! that the circuit can be turned on, and that the device firmware
$! can transfer data properly.  The data will be turned around in
$! the device line unit.  This sequence assumes that the circuit under
$! test is initially in the "OFF" state.
$!
$RUN $NCP
!
! These commands condition the line for the loopback test.
!
SET LINE DMR-0 CONTROLLER LOOPBACK
SET LINE DMR-0 VERIFICATION OFF
SET LINE DMR-0 DUPLEX FULL
SET CIRCUIT DMR-0 STATE ON
!
$EOD
$!
$! Since it may take several seconds for a circuit to complete changing
$! state from "OFF" to "ON", the following routine builds in a delay
$! of 5 seconds into the batch procedure.  This value may be modified
$! by editing this control file if it is not long enough.
$!
$RUN $NETSLP
5
$EOD
$!
$RUN $NCP
!
! This command displays the status of circuit DMR-0, to confirm that
! it is in the "ON" state.
!
SHOW CIRCUIT DMR-0 STATUS
!
! These commands define a temporary loopback nodename HTEST and
! perform the line-level loopback test.  A logical link will be created
! between the local NCP and MIRROR tasks. The "LOOP" command assumes
! that a default account has been defined for the MIRROR task at the
! local node; if not, accounting information should be supplied on the
! "LOOP" command line, with the parameters "USER" and "PASSWORD".
!
SET NODE HTEST CIRCUIT DMR-0
SHOW NODE HTEST COUNTERS
SHOW CIRCUIT DMR-0 COUNTERS
!
LOOP NODE HTEST COUNT 20
!
```

```
! These commands display node and circuit counters which can be compared
! with the "COUNT" parameter in the "LOOP" command. Also, circuit
! counters will indicate whether any data errors were detected.
!
SHOW NODE HTEST COUNTERS
SHOW CIRCUIT DMR-0 COUNTERS
!
! These commands restore the node and line status to the initial state
! expected by the cable loopback test below. If the cable loopback test
! is removed from this control file, and if the line was originally
! half-duplex, a command should be inserted here to return the line to
! half-duplex status (SET LINE DMR-0 DUPLEX HALF).
!
SET CIRCUIT DMR-0 STATE OFF
CLEAR NODE HTEST CIRCUIT
SET LINE DMR-0 CONTROLLER NORMAL
!
$EOD
$!
$! ***********************************************************************
$!
$! This section performs a cable loopback test, which verifies
$! that the circuit can be turned on, and that the device hardware
$! can transfer data properly. The data will be turned around by
$! a hardware turnaround connector or by a local modem; this configuration
$! must be set up MANUALLY before this test is run. This sequence assumes
$! that the circuit under test is initially in the "OFF" state.
$!
$! These commands condition the line for the loopback test.
$!
$RUN $NCP
!
SET LINE DMR-0 CONTROLLER NORMAL
SET LINE DMR-0 VERIFICATION OFF
SET LINE DMR-0 DUPLEX FULL
SET CIRCUIT DMR-0 STATE ON
!
$EOD
$!
$! Since it may take several seconds for a circuit to complete changing
$! state from "OFF" to "ON", the following routine builds in a delay
$! of 5 seconds into the batch procedure. This value may be modified
$! by editing this control file if it is not long enough.
$!
$RUN $NETSLP
5
$EOD
$!
$RUN $NCP
!
! This command displays the status of circuit DMR-0, to confirm that
! it is in the "ON" state.
!
SHOW CIRCUIT DMR-0 STATUS
!
! These commands define a temporary loopback nodename HTEST and
! perform the cable-level loopback test. A logical link will be
! created between the local NCP and MIRROR tasks.
!
SET NODE HTEST CIRCUIT DMR-0
SHOW NODE HTEST COUNTERS
SHOW CIRCUIT DMR-0 COUNTERS
!
LOOP NODE HTEST COUNT 20
!
```

```
! These commands display node and circuit counters which can be compared
! with the "COUNT" parameter in the "LOOP" command.  Also, circuit
! counters will indicate whether any data errors were detected.
!
SHOW NODE HTEST COUNTERS
SHOW CIRCUIT DMR-0 COUNTERS
!
! These commands restore the node and line status.  If the line was
! originally half-duplex, a command should be inserted to restore
! that status also (SET LINE DMR-0 DUPLEX HALF).  The hardware loopback
! connector or modem switch used to physically turn the data around must
! be removed MANUALLY.
!
SET CIRCUIT DMR-0 STATE OFF
CLEAR NODE HTEST CIRCUIT
!
$EOD
$!
$! ***********************************************************************
$!
$EOJ
```

A.4 Adjacent Phase III Node Verification Control File (NTESTA.CTL)

```
$JOB/CCL/NAME=NTESTA/NOLIMIT/ERROR:FATAL
$!
$! ***********************************************************************
$!
$!       NTESTA.CTL       V2.000
$!
$! Copyright (C) 1981 by Digital Equipment Corporation, Maynard, Mass.
$!
$! Edit history:
$!
$! 2.000   24-Nov-81        DECnet/E V2.0 Release              MLS
$!
$! ***********************************************************************
$!
$! This file is a prototype BATCH control file designed to verify that
$! a DECnet/E V2.0 node can be connected to an adjacent Phase III node
$! and that data can be transferred between the two nodes.
$! The user should modify this file as indicated by the comments
$! to define the node's individual configuration and the surrounding
$! network configuration.  See the DECnet/E Network Installation Manual
$! for further information on the use of this file.  The prototype
$! configuration below is used in all prototype installation test
$! command files in this manual.
$!
$! ***********************************************************************
$!
$! Prototype local node assumptions:
$!       ALPHA    = your node name, ie the executor node name
$!       1        = the executor node number
$!       DMR-0    = point-to-point device
$!       DMR-1    = point-to-point device (answers phone - slow line)
$!       DMP-0    = multipoint device (master)
$!       DMP-1    = multipoint device (slave)
$!
```

```
$! Prototype network node assumptions:
$!       BETA     =  2 = an adjacent Phase III node via DMR-0
$!       GAMMA    =  3 = an adjacent Phase II node via DMR-1
$!       DELTA    =  4 = an adjacent Phase III tributary via DMP-0
$!       EPSILN   =  5 = an adjacent Phase III tributary via DMP-0
$!       ZETA     =  6 = an adjacent Phase III master via DMP-1
$!
$! The test below uses circuit DMR-0 to node BETA.  This test should
$! be repeated for each adjacent Phase III node on each circuit.
$! Note that for multipoint circuits, no special procedures are necessary,
$! since only software-level loopback tests are used.
$!
$! ***********************************************************************
$!
$! This section contains NCP commands to perform external loopback tests,
$! using a communication line.  This will verify that the line can be
$! used to transfer data between the local node and the adjacent Phase III
$! node, and that compatible versions of network management are installed
$! on the two nodes. The "SHOW COUNTERS" commands display the number
$! of messages transferred; this should be compared with the "COUNT"
$! parameter in the "LOOP" command.
$!
$RUN $NCP
!
! This command turns on the circuit to node BETA.
!
SET CIRCUIT DMR-0 STATE ON
$!
$EOD
$!
$! Since turning on a circuit can take several seconds, due to the
$! node initialization process, the following program is run to provide
$! a delay of 10 seconds.  The number of seconds should be modified if
$! necessary; note that disk I/O is performed whenever node password
$! verification is performed.
$!
$RUN $NETSLP
10
$EOD
$!
$RUN $NCP
!
! This command displays the status of the circuit to node BETA.
! The display should confirm that the circuit is in the "ON" state
! and that the network software has initialized correctly with the
! adjacent node BETA.
!
SHOW CIRCUIT DMR-0 STATUS
!
! This command defines a temporary loopback node name RMLOOP for the
! circuit under test, and runs a loopback test between the local NCP
! and MIRROR tasks.  The data will be turned around by the Transport
! software on the remote node BETA to which the circuit is connected.
! Modify the command to insert the name of the adjacent node in place
! of "BETA", and the circuit-id of the communications device in place
! of "DMR-0".  The "LOOP" command assumes that a default account has
! been defined for the MIRROR task at the local node; if not, accounting
! data should be supplied on the "LOOP" command line, with the "USER"
! and "PASSWORD" parameters.
!
! NOTE:  This loop test will fail if node BETA is a Phase III
!        non-routing node.
!
SET NODE RMLOOP CIRCUIT DMR-0
!
```

```
SHOW NODE RMLOOP COUNTERS
SHOW CIRCUIT DMR-0 COUNTERS
SHOW NODE BETA COUNTERS
SHOW EXECUTOR COUNTERS
!
LOOP NODE RMLOOP COUNT 20
!
SHOW NODE RMLOOP COUNTERS
SHOW CIRCUIT DMR-0 COUNTERS
SHOW NODE BETA COUNTERS
SHOW EXECUTOR COUNTERS
!
CLEAR NODE RMLOOP CIRCUIT
!
! This command runs a loopback test between the local NCP and the
! MIRROR task at the remote node BETA.  Modify the commands to insert
! the name of the adjacent node in place of "BETA", and the circuit-id
! of the communications device in place of "DMR-0".  This command assumes
! that a default account has been defined for the MIRROR task on the
! node BETA; if not, accounting data should be supplied on the "LOOP"
! command line, with the "USER" and "PASSWORD" parameters.
!
LOOP NODE BETA COUNT 20
!
SHOW CIRCUIT DMR-0 COUNTERS
SHOW NODE BETA COUNTERS
SHOW EXECUTOR COUNTERS
!
$EOD
$!
$! *********************************************************************
$!
$! This section contains NFT commands to verify that the network file
$! access routines are installed where they are expected, and that a
$! file can be copied between nodes.  Modify the commands to insert
$! a valid user-id and password for BETA, to insert the name of your
$! adjacent node in place of "BETA", and to insure that a valid file
$! specification appears as the source for the copy operation.
$!
$! These commands create a logical link between the local NFT task
$! and the FAL task on the node BETA.
$!
$RUN $NFT
COPY BETA::NETTST.TMP=[1,2]NTESTA.CTL
[1,100]
DNETST

TYPE BETA::NETTST.TMP
DELETE BETA::NETTST.TMP;1
$EOD
$!
$! *********************************************************************
$!
$! This section turns the circuit to the adjacent node off again.
$!
$RUN $NCP
!
! This command returns the local node to its previous state, so that
! links to other adjacent nodes may be verified independently.
! Modify the command to insert the circuit-id of the communications
! device in place of "DMR-0".
!
```

```
SET CIRCUIT DMR-0 STATE OFF
!
$EOD
$!
$EOJ
```

A.5 Adjacent Phase II Node Verification Control File (NTEST2.CTL)

```
$JOB/CCL/NAME=NTEST2/NOLIMIT/ERROR:FATAL
$!
$! **********************************************************************
$!
$!      NTEST2.CTL      V2.000
$!
$! Copyright (C) 1981 by Digital Equipment Corporation, Maynard, Mass.
$!
$! Edit history:
$!
$! 2.000   24-Nov-81       DECnet/E V2.0 Release              MLS
$!
$! **********************************************************************
$!
$! This file is a prototype BATCH command file designed to verify that
$! a DECnet/E V2.0 node can be connected to an adjacent Phase II node
$! and that data can be transferred between the two nodes.
$! The user should modify this file as indicated by the comments
$! to define the node's individual configuration and the surrounding
$! network configuration.  See the DECnet/E Network Installation Manual
$! for further information on the use of this file.  The prototype
$! configuration below is used in all prototype installation test
$! command files in this manual.
$!
$! **********************************************************************
$!
$! Prototype local node assumptions:
$!      ALPHA   = your node name, ie the executor node name
$!      1       = the executor node number
$!      DMR-0   = point-to-point device
$!      DMR-1   = point-to-point device (answers phone - slow line)
$!      DMP-0   = multipoint device (master)
$!      DMP-1   = multipoint device (slave)
$!
$! Prototype network node assumptions:
$!      BETA    =  2 = an adjacent Phase III node via DMR-0
$!      GAMMA   =  3 = an adjacent Phase II node via DMR-1
$!      DELTA   =  4 = an adjacent Phase III tributary via DMP-0
$!      EPSILN  =  5 = an adjacent Phase III tributary via DMP-0
$!      ZETA    =  6 = an adjacent Phase III master via DMP-1
$!
$! The test below uses circuit DMR-1 to node GAMMA. This test should
$! be repeated for each adjacent Phase II node on each relevant circuit.
$!
$! **********************************************************************
$!
$! This section contains NCP commands to perform external loopback tests,
$! using a communication line.  This will verify that the line can be
$! used to transfer data between the local node and the adjacent Phase II
$! node, and that compatible versions of network management are installed
$! on the two nodes. The "SHOW COUNTERS" commands display the number
$! of messages transferred; this should be compared with the "COUNT"
$! parameter in the "LOOP" command.
$!
```

```
$RUN $NCP
!
! This command turns on the circuit to node GAMMA.
!
SET CIRCUIT DMR-1 STATE ON
!
$EOD
$!
$! Since a circuit may take several seconds to change its state from
$! "OFF" to "ON", the following program delays 10 seconds to allow the
$! state change to complete.  This time should be modified if insufficient.
$!
$RUN $NETSLP
10
$EOD
$!
$RUN $NCP
!
! This command displays the status of the circuit to node GAMMA.
! The display should confirm that the circuit is in the "ON" state
! and that the network software has initialized correctly with the
! adjacent node GAMMA.  (Note that in the prototype configuration
! above, the device DMR-1 is represented as answering a telephone
! line in order to make the connection.  If this is the case for your
! configuration, you and the remote operator at node GAMMA must
! co-operate to form the physical connection after each node has
! turned its circuit "on".  This may be done before starting this
! command file, since no error is generated by setting a circuit "on"
! when it is already "on".)
!
SHOW CIRCUIT DMR-1 STATUS
!
! This command runs a loopback test between the local NCP and the
! Phase II NICE looper task at the remote node GAMMA.  (If the node
! GAMMA is running DECnet/E V1.1, the NCU task will turn the data
! around.)  Modify the commands to insert the name of the adjacent
! node in place of "GAMMA", and the circuit-id of the communications
! device in place of "DMR-1".  The "LOOP" command assumes that a
! default account or some other procedure exists for the Phase II
! loopback program on node GAMMA.  If not, accounting information may
! be supplied on the "LOOP" command with the parameters "USER" and
! "PASSWORD".  For DECnet/E V1.1, this is not necessary.
!
SHOW CIRCUIT DMR-1 COUNTERS
SHOW NODE GAMMA COUNTERS
SHOW EXECUTOR COUNTERS
!
LOOP NODE GAMMA COUNT 20
!
SHOW CIRCUIT DMR-1 COUNTERS
SHOW NODE GAMMA COUNTERS
SHOW EXECUTOR COUNTERS
!
$EOD
$!
$! **********************************************************************
$!
$! This section contains NFT commands to verify that the network file
$! access routines are installed where they are expected, and that a
$! file can be copied between nodes.  Modify the commands to insert
$! a valid user-id and password for GAMMA, to insert the name of your
$! adjacent node in place of "GAMMA", and to insure that a valid file
$! specification appears as the source for the copy operation.
$!
```

```
$! These commands create a logical link between the local NFT task
$! and the FAL task on the node GAMMA.
$!
$RUN $NFT
COPY GAMMA::NETTST.TMP=[1,2]NTEST2.CTL
[1,100]
DNETST

TYPE GAMMA::NETTST.TMP
DELETE GAMMA::NETTST.TMP;1
$EOD
$!
$! ***********************************************************************
$!
$!  This section turns the circuit to the adjacent node off again.
$!
$RUN $NCP
!
! This command returns the local node to its previous state, so that
! links to other adjacent nodes may be verified independently.
! Modify the command to insert the circuit-id of the communications
! device in place of "DMR-1".
!
SET CIRCUIT DMR-1 STATE OFF
!
$EOD
$!
$EOJ
```

A.6 DTS/DTR Control File (DTSDTR.CTL)

```
$JOB/CCL/NAME=DTSDTR/NOLIMIT/ERROR:FATAL
$!
$! ***********************************************************************
$!
$!      DTSDTR.CTL      V2.000
$!
$! Copyright (C) 1981 by Digital Equipment Corporation, Maynard, Mass.
$!
$! Edit history:
$!
$! 2.000   24-Nov-81        DECnet/E V2.0 Release              MLS
$!
$! ***********************************************************************
$!
$! This file is a prototype BATCH command file designed to verify that
$! a DECnet/E V2.0 node can be connected to an adjacent Phase II node
$! and that data can be transferred between the two nodes.
$! The user should modify this file as indicated by the comments
$! to define the node's individual configuration and the surrounding
$! network configuration.  See the DECnet/E Network Installation Manual
$! for further information on the use of this file.  The prototype
$! configuration below is used in all prototype installation test
$! command files in this manual.
$!
$! ***********************************************************************
$!
```

```
$! ********************************************************************
$!
$!   This section turns the circuit to the adjacent node off again.
$!
$RUN $NCP
!
! This command returns the local node to its previous state, so that
! links to other adjacent nodes may be verified independently.
! Modify the command to insert the circuit-id of the communications
! device in place of "DMR-1".
!
SET CIRCUIT DMR-1 STATE OFF
!
$EOD
$!
$EOJ
```

A.7 Network Startup Command File (DECNET.CMD)

```
DETACH
LOGIN KB: [1,2]
FORCE KB: RUN $UTILTY
FORCE KB: SEND KB0: "Starting up Decnet/E V2.0"
FORCE KB:
FORCE KB: CCL TLK=
FORCE KB: CCL NCP=
FORCE KB: CCL NFT=
FORCE KB: CCL NET=
FORCE KB: CCL TLK-     =[1,2]TLK.*        ;PRIV 30000
FORCE KB: CCL NCP-     =[1,2]NCP.TSK      ;PRIV 30000
FORCE KB: CCL NFT-     =[1,2]NFT.TSK      ;PRIV 0
FORCE KB: CCL NET-WORK=[1,2]NET.TSK       ;PRIV 30000
FORCE KB: EXIT
FORCE KB:
FORCE KB: RUN $TTYSET
FORCE KB: WIDTH 80
FORCE KB: EXIT
FORCE KB:
FORCE KB: RUN $NCP
FORCE KB: !
FORCE KB: ! ******************************************************************
FORCE KB: !
FORCE KB: !        DECNET.CMD       V2.000
FORCE KB: !
FORCE KB: ! Copyright (C) 1981 by Digital Equipment Corporation, Maynard, Mass.
FORCE KB: !
FORCE KB: ! Edit history:
FORCE KB: !
FORCE KB: ! 2.000   24-Nov-81 DECnet/E V2.0 Release              MLS
FORCE KB: !
FORCE KB: ! ******************************************************************
FORCE KB: !
FORCE KB: ! This file is a prototype NCP command file designed to start up
FORCE KB: ! a DECnet/E V2.0 network.  It does not start any circuits, since it
FORCE KB: ! is designed for use in an installation checkout procedure, wherein
FORCE KB: ! circuits will be turned on and checked one at a time.  The file is
FORCE KB: ! designed to be used with the RSTS system startup procedure; see the
FORCE KB: ! RSTS/E SYSGEN Manual for a description of this startup procedure.
FORCE KB: ! See the DECnet/E Network Installation Manual for further information
FORCE KB: ! on the use of this file.  Note that this file assumes that the
```

```
FORCE KB: ! startup procedure is being run from the [1,2] account, and that the
FORCE KB: ! network tasks reside in the [1,2] account.  Edit this file, if
FORCE KB: ! necessary, to change the accounts to those specific to your system.
FORCE KB: !
FORCE KB: ! ******************************************************************
FORCE KB: !
FORCE KB: ! Prototype local node assumptions:
FORCE KB: !        ALPHA    = your node name, ie the executor node name
FORCE KB: !        1        = the executor node number
FORCE KB: !
FORCE KB: ! ******************************************************************
FORCE KB: !
FORCE KB: ! This command loads the volatile parameter file from the permanent
FORCE KB: ! parameter file. If no volatile parameter file exists, one will be
FORCE KB: ! created.  The volatile parameter file will be installed as a RSTS/E
FORCE KB: ! system file, and the event logger will be started if the NCP command
FORCE KB: ! "DEFINE ALL LOGGING STATE ON" was issued for the permanent parameter
FORCE KB: ! file.
FORCE KB: !
FORCE KB: SET SYSTEM
FORCE KB: !
FORCE KB: ! Any temporary changes to your network configuration or operating
FORCE KB: ! parameters can be made now with NCP "SET" commands; they will not
FORCE KB: ! affect the permanent parameter file.
FORCE KB: !
FORCE KB: ! ******************************************************************
FORCE KB: !
FORCE KB: ! This command starts network operation, ie, turns the network on.
FORCE KB: ! Internal data structures for Transport and NSP are initialized.
FORCE KB: !
FORCE KB: SET EXECUTOR STATE ON
FORCE KB: !
FORCE KB: ! ******************************************************************
FORCE KB: !
FORCE KB: ! This command starts the circuits which are in normal use.  Edit
FORCE KB: ! this file to change the names of the circuits to those used on
FORCE KB: ! your system.
FORCE KB: !
FORCE KB: SET CIRCUIT DMR-0 STATE ON
FORCE KB: SET CIRCUIT DMR-1 STATE ON
FORCE KB: SET CIRCUIT DMP-0.0 STATE ON
FORCE KB: !
FORCE KB: ! ******************************************************************
FORCE KB: !
FORCE KB: ! The following command starts NPKDVR such that it will run as a
FORCE KB: ! detached job and accept up to eight incoming network terminal
FORCE KB: ! connections.  If NPKDVR is not started this way, it will be
FORCE KB: ! spawned anew for each incoming connect into a separate job slot
FORCE KB: ! (assuming it has been defined as an object with an NCP DEFINE
FORCE KB: ! OBJECT command).
FORCE KB: !
FORCE KB: EXIT
FORCE KB:
FORCE KB: RUN $NPKDVR.TSK
ATTACH
```

Appendix B
Object Type Codes

Code	Type of Process
000	General Task, User Process
001	File Access (FAL/DAP Version 1)
002	Unit Record Services (URD)
003	Application Terminal Services (ATS)
004	Command Terminal Services (CTS)
005	RSX–11M Task Control, Version 1
006	Operator Services Interface
007	Node Resource Manager
008	IBM 3270-BSC Gateway
009	IBM 2780-BSC Gateway
010	IBM 3790-SDLC Gateway
011	TPS Application
012	RT–11 DIBOL Application
013	TOPS–20 Terminal Handler
014	TOPS–20 Remote Spooler
015	RSX–11M Task Control, Version 2
016	TLK Utility (LSN on DECnet/E)
017	File Access (FAL/DAP Version 4 and later)
018	RSX–11S Remote Task Loader
019	Network Management Listener (NICE Process)
020	RSTS/E Media Transfer Program (NETCPY)
021	Reserved for DECnet use
022	Mail Listener
023	Host Terminal Handler (NPKDVR)
024	Concentrator Terminal Handler
025	Loopback Mirror
026	Event Receiver
027	VAX/VMS Personal Message Utility
028	File Transfer Spooler (FTS)
029-062	Reserved for DECnet use
063	DECnet test tool (DTR)
064-127	Reserved for DECnet use
128-255	Reserved for customer extensions

Appendix C
Hardware Loopback Testing

The term *loopback* refers to the testing mechanisms in which data passes from its source, through a number of network components, and back to the source. If the data returns uncorrupted, that portion of the communications circuit is assumed to be functioning properly.

There are five levels of loopback testing possible with DECnet/E V2.0. Each level is characterized by the number of system components through which the data passes and the point in the circuit at which the data is turned around and returned to its source. These five levels are illustrated in Figure C–1 and are listed below:

1. **Local logical link loopback**. Local loopback occurs when a local program establishes a logical link with itself or with another program at the local node. Data is turned around at the local Transport layer.
2. **Controller loopback**. Controller loopback occurs when the communications controller (the DMC11, DMR11, DMV11 or DMP11) is placed in a special maintenance mode. Data is turned around within the controller itself, without involving the cable, modem (if any), or physical communications line.
3. **Hardware loopback**. Hardware loopback occurs when the data is turned around by a physical loopback device installed or enabled in the communications circuit, somewhere between the local controller and the remote system.
4. **Remote node loopback**. Remote node loopback occurs when the data is turned around by the remote Transport layer before reaching a network or user program on the remote node. (This type of loopback is only available on Phase III routing nodes.)
5. **Remote logical link loopback**. Remote logical link loopback occurs when a program on the local node establishes a logical link with a remote program designed to return data to its source. The data is turned around at either the remote network application layer, network management layer, or user layer.

NOTE

Some DECnet implementations include another level of loopback – referred to as "line" loopback. This type of loopback uses a special maintenance protocol (Maintenance Operation Protocol, or MOP) operating through the communications device itself. DECnet/E does not support this type of line loopback, since it does not support any of the MOP functions.

Proceeding through these levels of testing, the loopback point moves progressively further away from the source of data. Thus, each level tests for accurate data transmission one step further out from the local system, making it possible to isolate any problem to a particular hardware or software component of the communications circuit.

This appendix is concerned with loopback tests that test the communications hardware from the local controller to the remote end of the physical communications line. Section C.1 describes the NCP command sequences necessary to initiate the tests. The actual hardware configurations required to test a communications circuit in point-to-point mode and in multipoint mode are discussed in Sections C.2 and C.3, respectively.

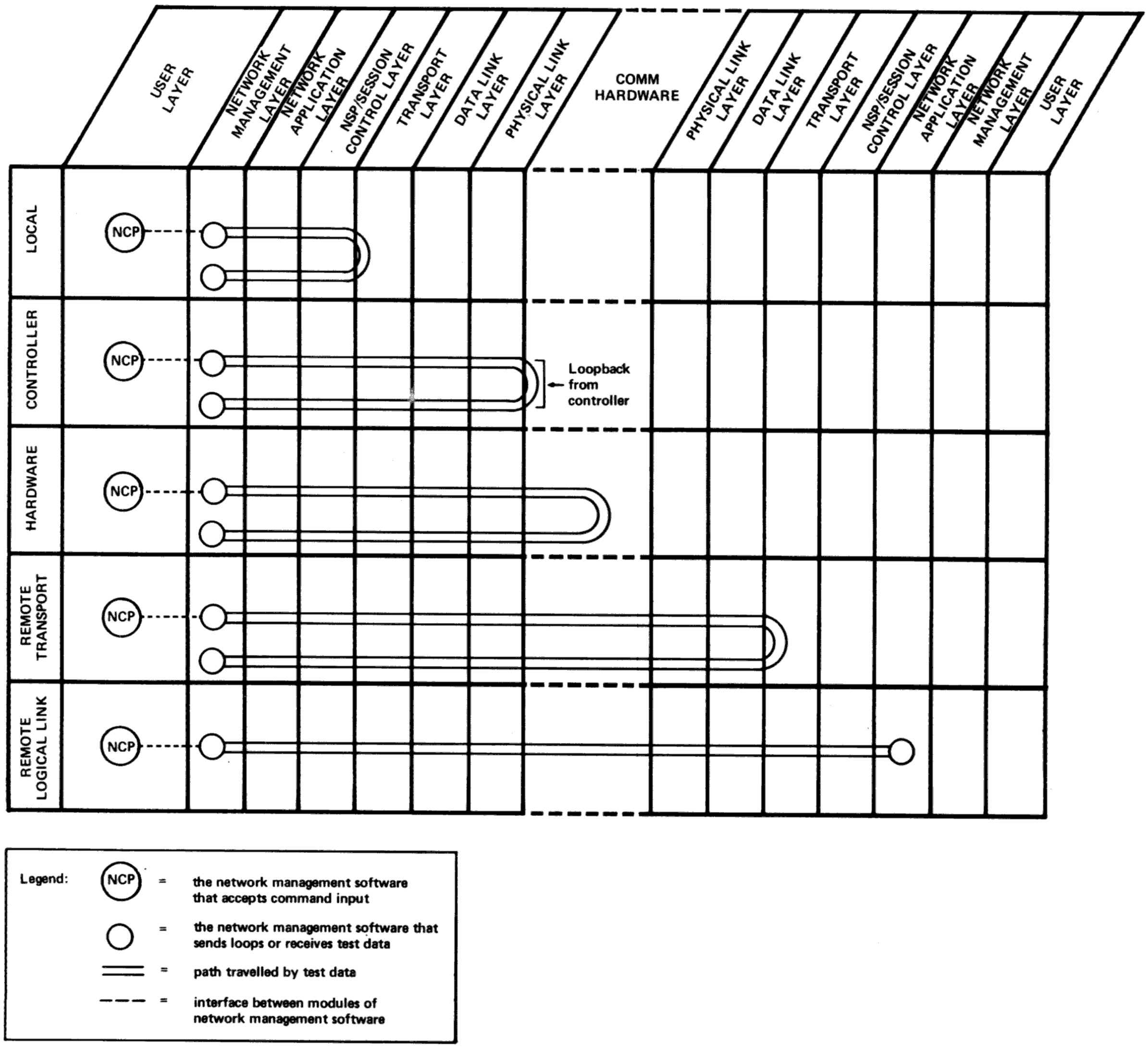


Figure C–1: Levels of Loopback Testing

C.1 Loopback Command Sequences

Controller-level loopback tests use a special maintenance feature of the communications controller to loop back data within the device itself. Other hardware components are not involved and there is no need to install additional loopback connectors or to disconnect cables or modems. The device must, however, operate in full-duplex mode. (See Figure C–2.)

To enable the controller maintenance function and run the loopback test, type the following NCP command sequence:

SET CIRCUIT *circuit-id* STATE OFF
SET LINE *line-id* DUPLEX FULL
SET LINE *line-id* CONTROLLER LOOPBACK
SET NODE *node-id* CIRCUIT *circuit-id*
SET CIRCUIT *circuit-id* STATE ON
LOOP NODE *node-id* *<options>*

The *circuit-id* identifies the communications line being tested, the *line-id* identifies the controller, and *node-id* is the temporary loopback node name being assigned to the loopback circuit. See the *DECnet/E System Manager's Guide* for the syntax of the options that can be specified with the LOOP NODE command.

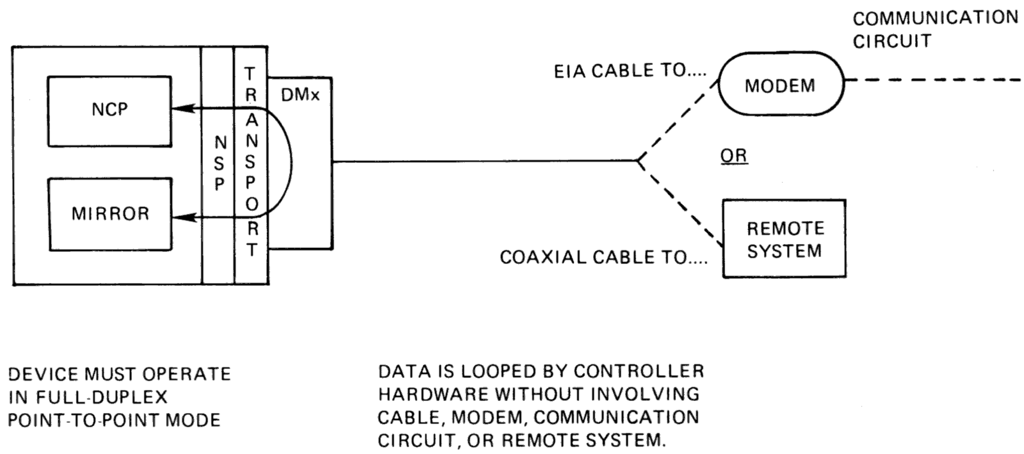


Figure C–2: Controller-Level Loopbacks

Hardware-level loopback tests operate by installing or enabling a loopback device at some point in the communications circuit. Once the hardware is configured properly (as described in Sections C.2 and C.3), the line must be *conditioned* (that is, established with network management as a loopback circuit) before the actual test can be run.

To condition the line and run the loopback test, type the following NSP command sequence:

SET CIRCUIT *circuit-id* STATE OFF
SET LINE *line-id* DUPLEX FULL
SET LINE *line-id* CONTROLLER NORMAL
SET NODE *node-id* CIRCUIT *circuit-id*
SET CIRCUIT *circuit-id* STATE ON
LOOP NODE *node-id* *<options>*

Once again, the *circuit-id* identifies the communications line, the *line-id* identifies the controller, and *node-id* is the temporary loopback node name assigned to the loopback circuit. See the *DECnet/E System Manager's Guide* for the syntax of the options that can be specified with the LOOP NODE command.

Data directed to a program on the loop node passes through the controller and out to the loopback point. At the loopback point it is turned around, returned through the controller, and delivered to the target program for analysis. The target program actually runs on the local node. Setting a pseudo-node name to the circuit under test tells the Transport module to route all traffic addressed to that node name onto the specific circuit and to accept the looped data as input to the local node.

C.2 Point-to-Point Hardware-Level Loopback

This section discusses the hardware arrangements necessary to do hardware-level loopback tests on DMC11/DMR11 controllers and on DMV11/DMP11 controllers set up for point-to-point communication. In terms of loopback testing, there is only one main difference between the various controllers. Typically, the DMR11, DMV11, and DMP11 have a connector/distribution panel between the controller and the actual communications line; the DMC11 does not. Thus, circuits operating with a DMC11 contain one less component to be tested.

NOTE

> Since the DMR11, DMV11, and DMP11 all have similar hardware configurations, for the rest of this section the term DMR11 will be used to refer to all three of these controller types.

The hardware arrangements necessary to do hardware-level loopback tests depend on the type of connecting cables used with the controller. In general, there are two types of cable – ribbon (EIA) cable and coaxial (pigtail) cable. The following subsections discuss the hardware arrangements required for testing circuits involving each type.

C.2.1 Ribbon Cable

Ribbon cable is generally used when the physical path to a remote system involves either switched (dial-up) or leased (private) telephone lines. Typically, such a communications circuit covers an extended distance and includes modems at both the local and remote ends of the line.

Diagrams A through E in Figure C–3 show the hardware configuration for the two types of controllers when ribbon cable is used. In both cases, one end of the cable connects to the controller's line unit board. With the DMC11, the other end of the cable terminates in a standard EIA connector that plugs directly into the local modem (see Diagram A). With the DMR11, on the other hand, the cable terminates at a connector panel, with a second cable connecting the panel and the modem (see Diagram B). The actual panel used depends on the type of cable. (Table C–1 lists some common cable interfaces and their appropriate panels.) Diagram C shows the most frequently installed panel, for use with RS-232-C and RS-423-A interface cables, and Diagram D shows the panel used with RS-422-A interface cables. Diagram E shows the special case DMR11 cabling used with V.35 interface modems that eliminates the connector panel completely.

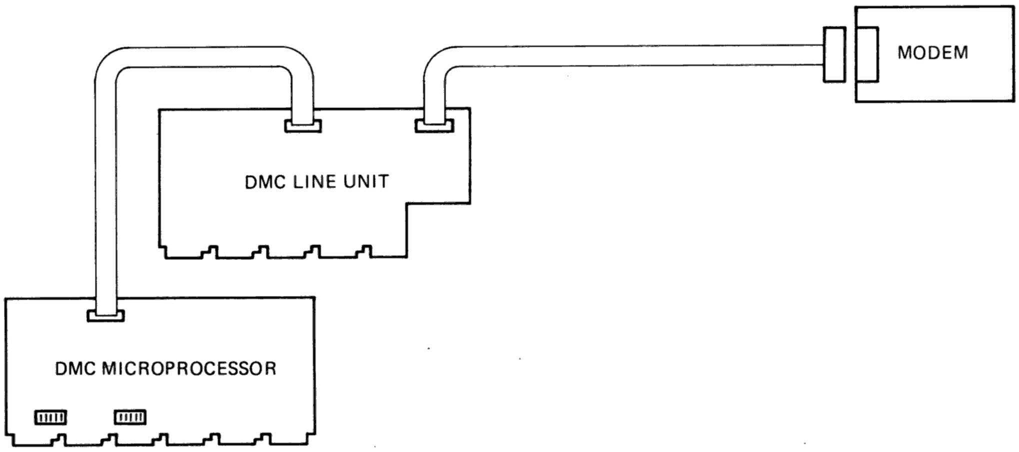


Figure C–3A: DMC11 Controller with Ribbon Cable

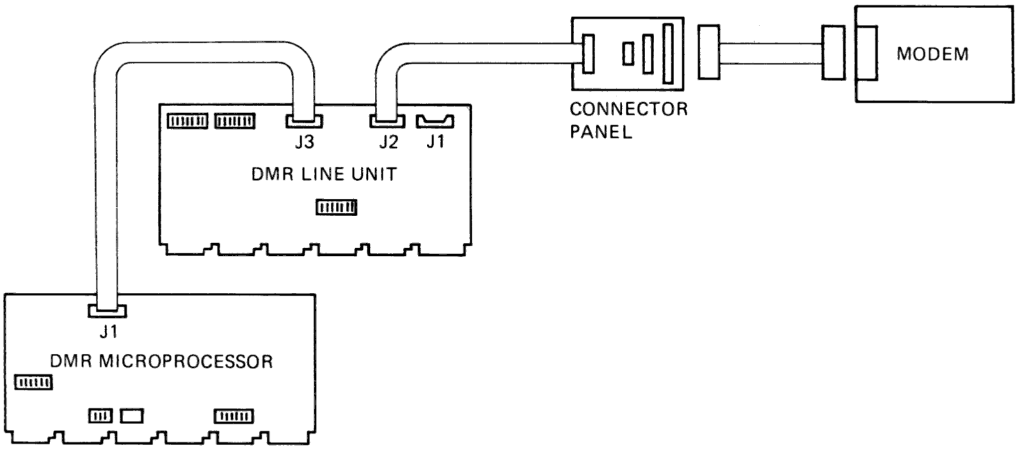


Figure C–3B: DMR11 Controller with Ribbon Cable

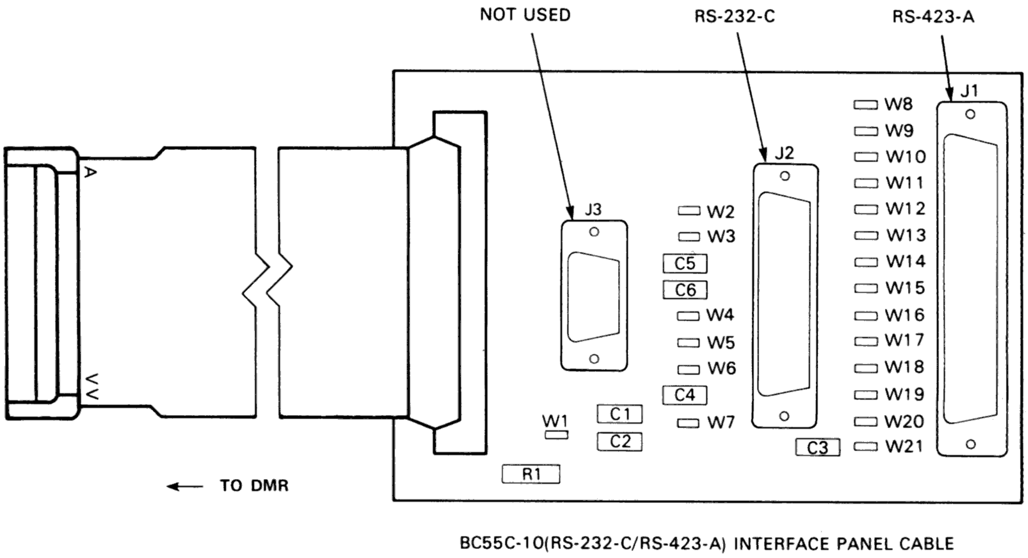


Figure C–3C: BC55C-XX (RS-232-C/RS-423-A Interface) Panel Cable

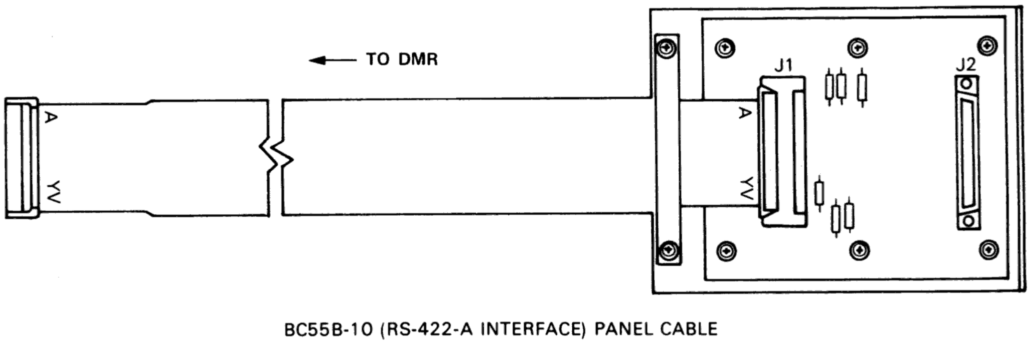


Figure C–3D: BC55B-XX (RS-422-A Interface) Panel Cable

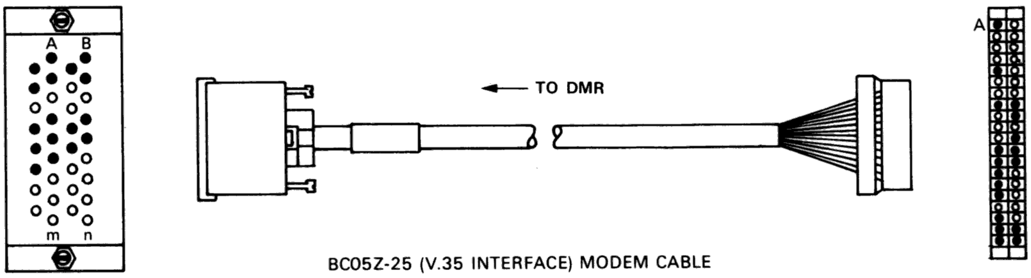


Figure C–3E: BC05Z-XX (V.35 Interface) Modem Cable

Loopback Connectors

Testing the hardware components from the controller to the local modem requires the installation of a loopback test connector at some point in the communications path. The type of connector used depends on the type of cable. Table C–1 lists the connector types for the various cable interfaces.

To test the communications path up to the connector/distribution panel (if any), the loopback connector is installed in the appropriate jack in the panel (as listed in Table C–1), the line is "conditioned" with the NCP command sequence given in Section C.1, and the loopback test is run.

To test the remainder of the path up to the local modem, the cable is unplugged from the modem and the loopback connector is installed in the connector at the end of the cable. The line is conditioned and the test is run, as before.

Table C–1: Cable Panels and Test Connectors

Cable Interface	Cable Panel	Loopback Connector	Panel Test Jack
RS-232-C	BC55C	H325	J2
RS-422-A	BC55B	H3251	J2
RS-423-A	BC55C	H3251	J1
V.35	BC05Z	H3250	–

Local Modem Loopback

To test the communications path out to, and including, the local modem, a special maintenance feature of the modem is used. A common option is an analog loopback feature that directs the modem's transmitter output to its receiver input. This feature is usually activated by a switch, such as the analog loopback (AL) button available on Bell 208A and 208B modems.

NOTE

> The discussions of modem loopback features, here and in the following subsection, are based on Bell 200 series modems manufactured by Western Electric. Almost all modems have loopback and self-test options or features. Since hardware implementations differ among the various manufacturers, the precise details for doing loopback tests can differ from what is described here. Consult the operation manual for your modems regarding loopback features before using the procedures given here.

The modem converts digital data from the controller to an analog signal that would normally be transmitted over the communications line. With the analog loopback feature enabled, however, the transmitter output (the analog signal) is fed back into the modem receiver. (See Figure C–4.) The receiver digitizes the analog signal and returns the digital data to the controller.

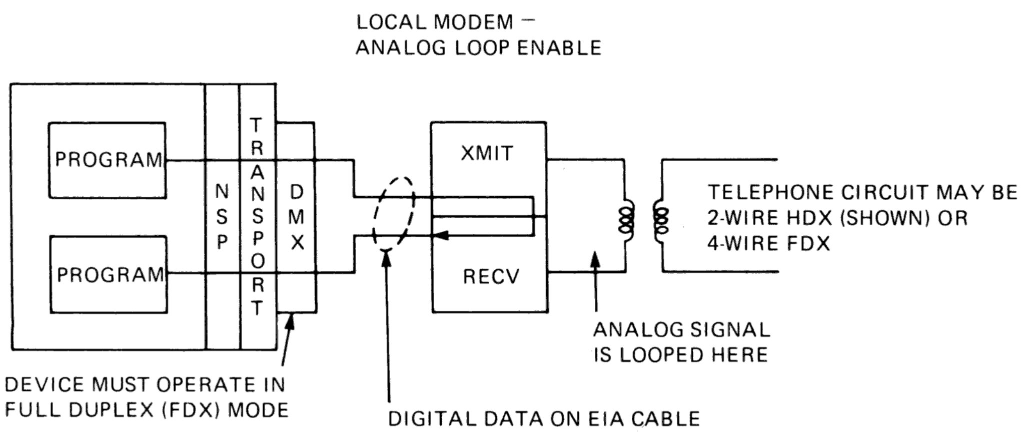


Figure C–4: Local Modem Loopback

The controller hardware requires Data Set Ready to be on (asserted) before it will transmit data, but many modems clear this signal when analog loop is enabled. Bell 208A, 208B, and 209 modems offer a strapping option that determines whether the Data Set Ready signal is asserted or negated when the modem is in the analog loop test mode. Bell refers to this as the "CC Indication of Analog Loop (ON/OFF)" option.

If your modem clears Data Set Ready, but does not offer a strapping option such as described above, the local modem loopback test cannot be run without the use of a "breakout box" between the controller and the modem. A schematic of a typical breakout box, and the connections required to perform an analog loop test with a modem that clears Data Set Ready, are shown in Appendix D, Figure C–1.

To configure a communications circuit for local modem loopback, remove any loopback connectors installed for previous tests and reconnect the cable to the modem. Disconnect (hang up) the telephone line to the remote system and enable analog loopback on the local modem. Use the command sequence given in Section C.1 to condition the line as a loopback circuit and run the test. The controller must operate in full-duplex mode, even though the modem can be connected to a 2-wire (half-duplex) telephone line. Since the telephone circuit is not used, the type of line does not affect the operation of the test.

Remote Modem Loopback

The communications path can be tested up to and including the remote modem by using another common maintenance feature of the modem. This option, known as digital loopback, directs the modem's receiver output to its transmitter input. The feature is generally activated by a switch, such as the digital loopback (DL) button available on the Bell 208 modems.

In this case, the local modem operates normally in that it converts digital data from the controller to an analog signal and transmits the signal over the communications line. The remote modem's receiver converts the analog signal to digital form and loops the digital data back to its transmitter. The remote modem's transmitter converts the digital data to an analog signal and transmits the analog signal back over the telephone circuit. The local modem digitizes the data once again and returns it to the controller. (See Figure C–5.)

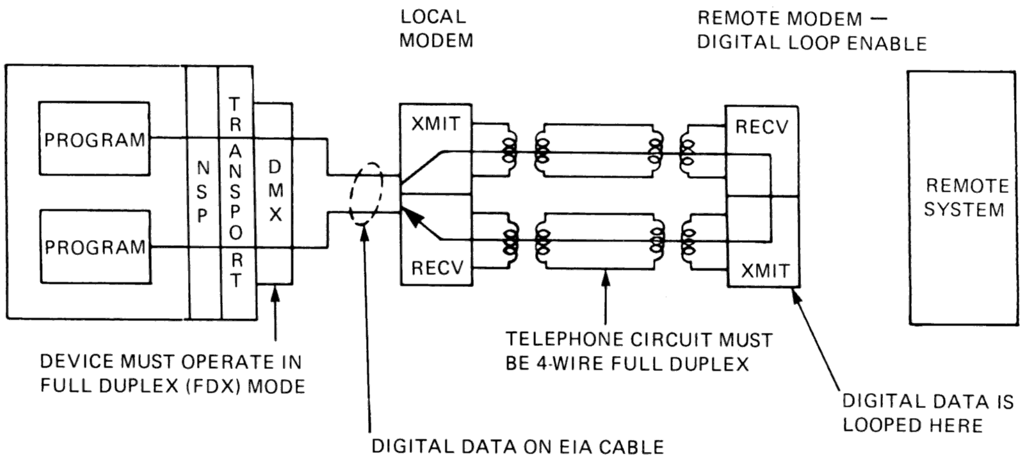


Figure C–5: Remote Modem Loopback

To configure a communications circuit for remote modem loopback, ask the remote operator to disable the line interface connected to the remote modem and to enable the digital loopback feature. (Some modems have a remote digital loop option that allows a local operator to switch the remote modem into digital loopback without the intervention of an operator at the remote site.) Next, condition the communications line as a loopback circuit using the NCP command sequence given in Section C.1. Establish the telephone connection to the remote modem and run the loopback test.

Unlike the other tests described so far, the type of communications line *does* matter for this type of loopback test. The entire circuit (including both modems and the communications lines themselves) must be capable of full-duplex operation. Switched (dial-up) telephone lines are 2-wire circuits. Although there are several modems that operate full-duplex over 2-wire circuits, a 2-wire line usually implies half-duplex operations. In some circumstances, dual-dial arrangements can be used to create a 4-wire full-duplex channel from two switched lines. Leased (private) lines can be either 2- or 4-wire circuits. Basically, if your installation is not set up for full-duplex communication to the remote system, you cannot perform remote modem loopback.

C.2.2 Coaxial Cable

Coaxial (pigtail) cable is generally used when the physical path to a remote system covers only a short distance. Telephone circuitry is not involved and no modem is required.

Figure–6 shows the normal hardware configuration for the two types of controllers when coaxial cable is used. In the case of the DMC11, two coaxial cables connect to the line unit board and extend through the opening at the back of the mounting box. The cables are approximately one foot long and terminate in 4-pin female connectors (see A in, Figure C–6).

With the DMR11, on the other hand, a cable connects the line unit board to a connector/distribution panel (see B, in Figure C–6). The connector panel has four 4-pin connectors – two male and two female (see C, in Figure C–6). In a point-to-point configuration, only two of these connectors are used and have cables attached. The other two are fitted with 75 ohm terminators.

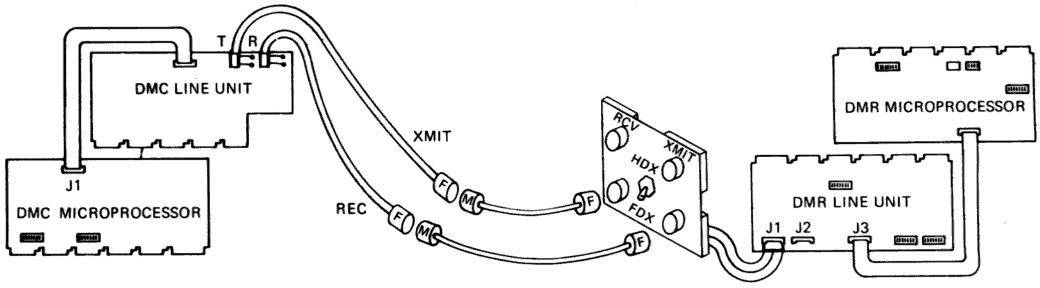


Figure C–6A: DMC11 Controller with Coaxial Cable

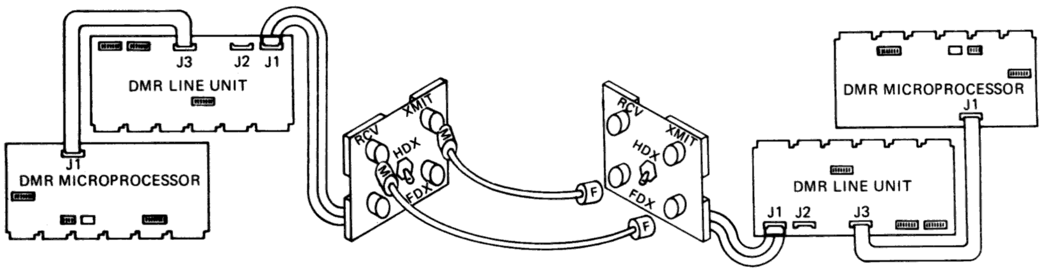


Figure C–6B: DMR11 Controller with Coaxial Cable

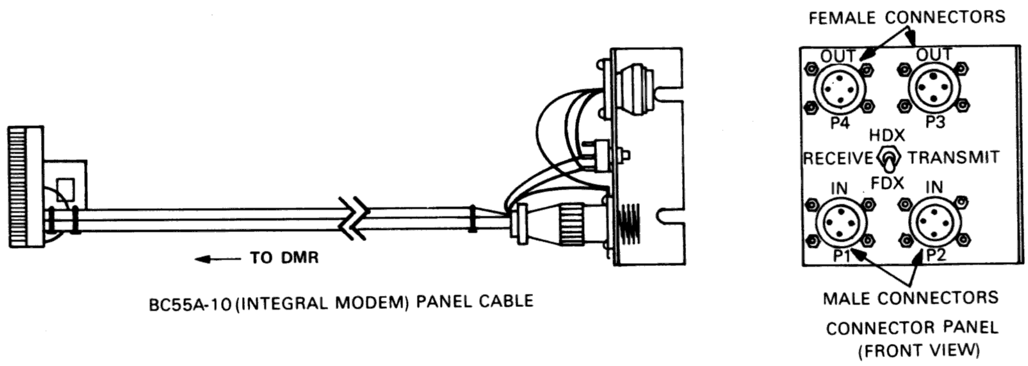


Figure C–6C: Coaxial Connector Panel

To test the communications path from the DMR11 to the connector panel, the duplex switch in the center of the panel is set to half-duplex (HDX). This causes the outgoing data to be turned around at the panel and returned to the controller. In this case, the data is also transmitted out over the cable. Thus, to prevent the remote system from mistakenly responding to the data sent and corrupting the test results, either the outgoing cables should be physically disconnected from the panel, or the circuit should be turned off (at the *remote* system) using the SET CIRCUIT STATE OFF command, or the NCP command that is apropriate at the remote node. When the necessary hardware adjustments have been made, the line is conditioned and the test is run, using the NCP command sequence given in Section C.1.

To test the cables to the remote system — either from the DMC11 line unit board or from the DMR11 connector panel — the cables are disconnected (at the remote end) and a loopback device is installed, connecting the ends of the two cables. The device used depends on the connector type on the end of the cables. If the cables terminate in female connectors, a small feed-through connector (12-12528) can be used (see Figure C–7). If the cables terminate in male connectors, a short (one foot) cable, fitted with two female connectors, should be used.

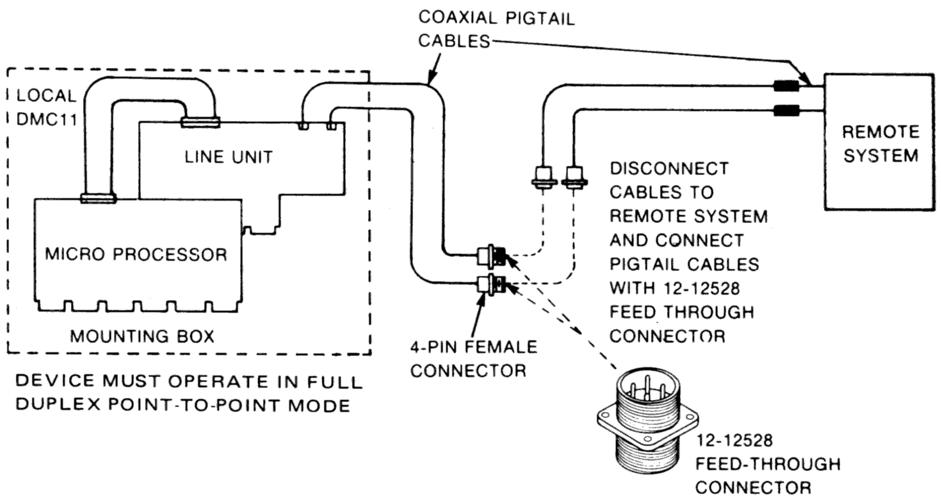


Figure C–7: Coaxial Cable Loopback Connector

NOTE

If the line to the remote system consists of more than one physical piece of cable, both types of loopback devices and several separate loopback tests may be necessary, along the length of the line, to completely test the line.

If the remote controller is a DMR11-type device (as shown in Diagrams A and B of Figure C–6), the remote connector panel itself can be used as the loopback device. To do this, the duplex switch in the center of the remote

panel is set to half-duplex (HDX). Incoming data is then turned around by the panel and sent back over the line to the local controller. The data is also transmitted through the remote connector panel to the remote controller. Thus, once again, to prevent the remote system from responding to the data and corrupting the test results, the circuit should be turned off (at the *remote* system) using the SET CIRCUIT STATE OFF command, or the NCP command that is appropriate.

When testing the line with a DMR11-type controller, no matter which hardware arrangement is used, be certain to reset the duplex switch on the local connector panel to the full-duplex position (FDX).

Once the necessary hardware adjustments have been made, the line is conditioned and the loopback test is run, using the NCP command sequence given in Section C.1.

NOTE

> Note that the DMR11 circuit shown in Diagram B of Figure C–6 is configured for full-duplex operation. Hardware-level loopback tests beyond the connector/distribution panel can only run with a full-duplex circuit. If you have a DMR11-type controller configured for half-duplex, it must be altered to full-duplex operation before running these tests.

C.3 Loopback Tests in a Multipoint Environment

In general, hardware loopback tests — either at the device level or the line level — are not really possible in a multipoint environment. To do installation testing of multipoint hardware or to isolate a problem in a running multipoint chair, the hardware must be reduced to a point-to-point configuration.

Installation testing has already been discussed in the previous section. Each hardware component is simply tested in point-to-point mode, and when all components function satisfactorily, they can be cabled together in a multipoint configuration. Isolating a problem in a *running* multipoint environment, however, does have some side effects that should be mentioned.

If a problem is suspected in a particular controller, the system to which that device is attached must be isolated from the rest of the multipoint chain. The operating protocol of the controller is then changed to point-to-point. This is done with the following NCP command sequence:

SET CIRCUIT *circuit-id* STATE OFF
SET LINE *line-id* PROTOCOL DDCMP POINT

(Note that the protocol of a controller cannot be changed while the circuit is operational. Further, to change the protocol of the control station, *all* circuits from the controller must be turned off.)

After that, controller-level loopback tests can be run, using the command sequence given in Section C.1. No cables need to be disconnected, and assuming the affected station is not the control station, running the loopback test in this fashion has no effect on the remainder of the multipoint chain. If the affected station is the control station, however, this procedure will destroy the entire multipoint environment unless another station can be designated as the control station.

If a problem is suspected in the portion of the circuit from a particular controller to its distribution panel, the station must be *physically* isolated from the chain. The cables must be disconnected from the front of the distribution panel and connected together, bypassing the station altogether. (Be certain to connect "Transmit In" to "Transmit Out" and "Receive In" to "Receive Out.")

After the station has been isolated in this fashion, and the controller's operating protocol has been changed to point-to-point (with the command sequence shown above), hardware-level loopback tests can be run between the controller and the panel. Once again, assuming that the affected station is not the control station, this procedure has no effect on the remainder of the multipoint chain.

If a problem is suspected in the hardware between two adjacent stations, however, *both* stations must be isolated from the chain and a point-to-point circuit must be established between them. This procedure will usually significantly disrupt the multipoint environment.

Consider the 5-station multipoint chain shown in Figure–8. To run hardware- or node-level loopback tests between tributaries C and D, those two stations must be isolated from the rest of the chain. Their operating protocols must be changed to point-to-point and the hardware must be configured so that there is a direct (point-to-point) circuit between them. To prevent interference from the control station (B), the cables between stations B and C must be disconnected (see Diagram B, in Figure C–8). Under these circumstances, the control station is no longer able to communicate with tributary E (although it is still able to communicate with tributary A). In fact, *no* station can communicate with tributary E. (Note that there is no need to disconnect the cables between stations D and E, since tributary E will not respond to any transmission other than a poll from the control station.)

Thus, before running controller- or hardware-level loopback tests in this fashion, consideration must be given to the effect on the total multipoint environment.

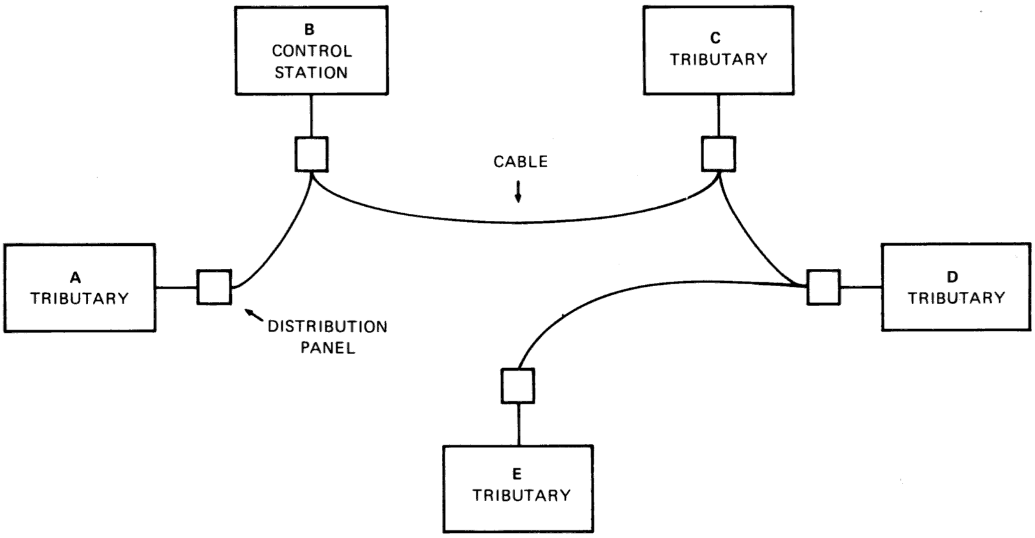


Figure C–8A: All Stations Running Multipoint

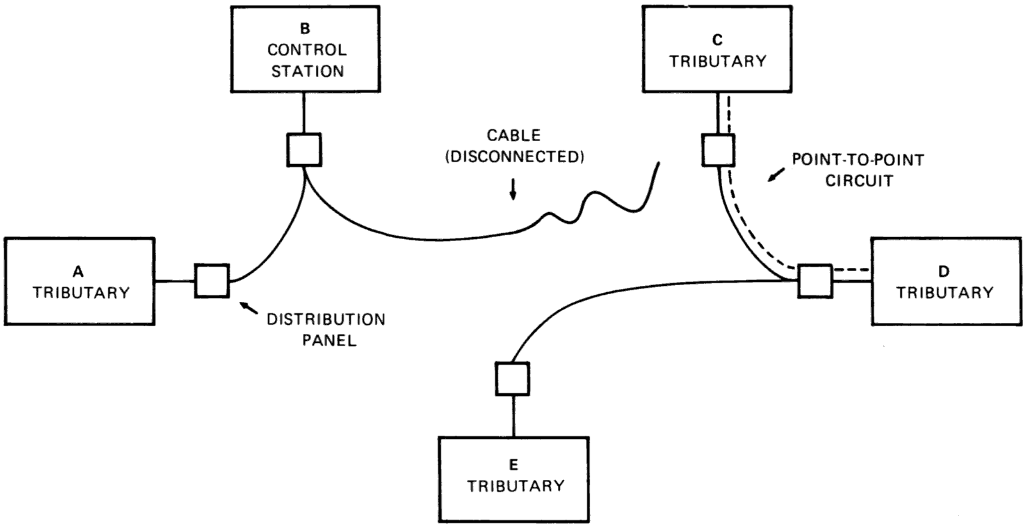


Figure C–8B: Stations C and D Running Point-to-Point

Appendix D
EIA/CCITT Interface Notes

D.1 EIA/CCITT Interface Notes

In the early days of data communications in the United States, the associated operating companies of American Telephone and Telegraph were, for all practical purposes, the only providers of data communications services. Therefore, the modems developed by the engineers at Bell Laboratories and manufactured by Western Electric were the standards of the industry. As time went on, computer and terminal equipment manufacturers needed to know the electrical characteristics of Bell modem interfaces. Similarly, as independent companies began to offer competing modems, they needed to know the electrical characteristics of the interfaces offered by the computer and terminal manufacturers.

To avoid chaos, the Electrical Industry Association (EIA), in cooperation with the Bell System, the independent modem manufacturers, and the computer manufacturers, developed a standard for the interface between Data Terminal Equipment (DTE) – computers and terminals – and Data Communication Equipment (DCE) – modems – employing Serial Binary Interchange. This standard is called RS-232, or more specifically RS-232-C, reflecting the latest (C) revision. Newer standards, RS-422 and RS-423, have been developed that will eventually replace the RS-232-C standard, but RS-232-C is still the standard to which modem interfaces are designed.

In countries other than the United States, data communication services are usually provided by government Post, Telegraph, and Telephone (PTT) authorities. To permit the economic manufacture of modems, terminals, and computers, and to facilitate data communications between various countries, the PTTs of the United Nations countries, in cooperation with the Comite Consultatif International Telephonique et Telegraphique (CCITT), also developed standards for data communication interfaces. Due to varying requirements, nationalism, and other factors that preclude firm standards, CCITT working committees develop recommendations that are adopted and published by the CCITT as a whole. These recommendations are numbered and published in books. The current volume relating to data communication is Volume VIII which contains recommendations prefixed with the letters V and X. CCITT recommendation V.24 is similar to the EIA RS-232-C standard in the USA.

Some of the information in this appendix was reproduced from "Technical Aspects of Data Communications" by John E. McNamara, published by the Educational Services Department, Digital Equipment Corporation.

Returning to the EIA standard, it is important to note exactly what the standard contains.

- The electrical signal characteristics (voltage, currents, impedances, and so forth).
- Mechanical characteristics of the interface (connector and pin assignments information).
- A functional description of the interface circuits.
- A list of standard subsets of specific interchange circuits for particular applications.

A brief description of the EIA/CCITT interface is provided in Table D-1.

Table D–1: EIA/CCITT Interface Summary

PIN	Direction	Circuit EIA/CCITT	Mnemonic Name	Description
1		AA/101	FG	*Protective (Frame) Ground.* This conductor is an electrical equipment frame ground. It can be further connected to external grounds as required by applicable regulations.
2	→DCE	BA/101	TD	*Transmitted Data.* This conductor carries the serial digital data transmitted from the DTE (computer or terminal) to the DCE (modem). The DCE then converts the digital data to a form that can be sent over the communication facilities to the remote system.
3	DTE ←	BB/104	RD	*Received Data.* Signals on this conductor are generated by the receiving DCE (modem) in response to data received from the remote system. This is the digital data sent from the remote system to the local equipment.
4	→ DCE	CA/105	RTS	*Request to Send.* The ON condition of this lead causes the modem to assume the data transmit mode.
5	DTE ←	CB/106	CTS	*Clear to Send.* Signals on this circuit indicate whether or not the modem is ready to transmit data. The ON condition of this lead, coupled with ON conditions on circuits CA, CC, and CD (when implemented) indicate that data presented on circuit BA (transmitted data) will be transmitted to the remote system.

(continued on next page)

Table D–1 (Cont.): EIA/CCITT Interface Summary

PIN	Direction	Circuit EIA/CCITT	Mnemonic Name	
6	DTE ←	CC/107	DSR or MR	*Data Set Ready* or *Modem Ready*. This circuit indicates the status of the local modem. An ON condition indicates that the local modem is ready to process data (modem is connected to the communication circuit; call is established; all handshaking has been completed; modem is not in test, talk, or dial mode).
7		AB/102	SG	*Signal Ground*. This conductor establishes the common reference potential for all interchange circuits except circuit AA (protective ground).
8	DTE ←	CF/109	DCD or CD	*Data Carried Detect (Received Line Signal Detector)*. The ON condition of this lead indicates that the DCE (modem) is receiving a signal (from the remote system) that meets the suitability criteria established by the modem manufacturer (that is, a signal the modem can understand).
9	DTE ←			*Positive DC Test Voltage*. Used for testing only.
10	DTE ←			*Negative DC Test Voltage*. Used for testing only.
11				Not assigned by RS-232-C.
	DTE ←	BELL 208A	QM	*Equalizer Mode*.
12	DTE ←	SCF/122	(S)DCD	*Secondary Data Carrier Detect*. (Secondary Received Line Signal Detector). This line performs the same function as circuit CF (Data Carrier Detect) except that it indicates proper reception of a signal on the secondary channel rather than the primary channel.
13	DTE ←	SCB/121	(S)CTS	*Secondary Clear to Send*. This signal performs the same function as circuit CB (Clear to Send) except that it indicates availability of the secondary channel rather than the primary channel.
14	→ DCE	SBA/118	(S)TD	*Secondary Transmitted Data*. This signal performs the same function as circuit BA (Transmitted Data) except that it carries data from the DTE to the modem for transmission on the secondary channel *rather than the* primary channel.

(continued on next page)

Table D–1 (Cont.): EIA/CCITT Interface Summary

PIN	Direction	Circuit EIA/CCITT	Mnemonic Name	
	→ DCE	BELL 208A	NS	*New Sync.*
15	DTE ←	DB/114	TC	*Transmitter Clock (Transmitter Signal Element Timing – DCE Source).* This circuit is used to provide signal element timing information to the DTE. The DTE provides a data signal on circuit BA (Transmitted Data) in which the transitions between signal elements (bits) nominally occur at the time of the OFF to ON transition of this lead. Used by the DTE (computer or terminal) to determine when it should present the next data bit on the transmitted data lead.
16	DTE ←	SBB/119	(S)RD	*Secondary Reveived Data.* This line performs the same function as circuit BB (Received Data) except that it carries data received by the modem on the secondary channel.
	DTE ←	BELL 208A	DCT	*Divided Clock, Transmitter.*
17	DTE ←	DD/115	RC	*Receiver Clock (Receiver Signal Element Timing – DCE Source).* This lead provides the DTE with received signal timing information. The ON to OFF transition of this signal nominally occurs at the center of each signal element (bit) on circuit BB (Received Data). Used by the DTE to determine when it should read data from the Received Data Line.
18				Not assigned by RS-232-C
	DTE ←	BELL 208A	DCR	*Divided Clock, Receiver.*
19	→ DCE	SCA/120	(S)RTS	*Secondary Request to Send.* This signal performs the same function as circuit CA (Request to Send) except that it requests use of the secondary channel rather than the primary channel.
20	→ DCE	CD/108.2	DTR	*Data Terminal Ready.* Signals on this lead are used to control switching of the DCE to the communication channel. The ON condition prepares the DCE for connection and maintains a connection established by external means such as manual call origination, manual answering, or automatic call origination.

(continued on next page)

Table D–1 (Cont.): EIA/CCITT Interface Summary

PIN	Direction	Circuit EIA/CCITT	Mnemonic Name	
21	DTE ←	CG/110	SQ	*Signal Quality Detect.* This circuit is used to indicate to the DTE whether or not there is a high probability of errors in the received data. The ON condition is maintained when there is no reason to believe that an error has occurred. An OFF condition indicates that there is a high probability of error.
22	DTE ←	CE/125	RI	*Ring Indicator.* The ON condition of this circuit indicates that a ringing signal is being received on the communication channel. In switched network applications the DTE (computer) would be expected to answer the call by asserting circuit CD (Data Terminal Ready).
23	→ DCE	CH/11		*Data Rate Selector (DCE Source).* This signal is used to select between two data signalling rates in the case of a dual rate synchronous modem. The ON condition selects the higher data rate.
	DTE ←	CI/112		*Data Rate Selector (DCE Source).* This signal is used to select one of two data signalling rates in the case of dual rate synchronous modems. The ON condition selects the higher data rate.
24	→ DCE	DA/113	(TC)	*External Transmitter Clock (Transmitter Signal Element Timing – DTE Source).* This circuit is used to provide the transmitting signal converter (modem transmitter) with signal element timing information. The ON to OFF transition of this lead nominally corresponds to the center of each element (bit) on circuit BA (Transmitted Data).
25				Not assigned by RS-232-C.
	→ DCE	BELL 208A		*Busy.* Used for testing.

D.2 Breakout Box Application Notes

Several manufacturers of data communication equipment market a piece of test equipment called a "breakout box" (or similar name). The purpose of the box is to allow patching and monitoring of the various leads of the EIA/CCITT interface. A schematic of a typical breakout box is shown in Figure D–1.

As shown in Figure D–1, the typical breakout box consists of two 25-pin EIA connectors, switches to break any of the signal lines, test/patch points to allow special interconnections, and, in many cases, a set of indicator lights for monitoring signals. The connector on the left in the drawing is a female EIA connector that is connected to a terminal or computer (DMC11 or other line interface hardware) with an EIA cable (see Figure D–2). The connector on the right is a male connector that plugs into a modem.

With all switches closed as shown in Figure D–1, the breakout box has no effect on the interface signals. If indicator lights are provided, they can be used to monitor common control signals (RTS, CTS, DSR, DCD, DTR, SQ, and RI) or Transmitted Data (TD) and Received Data (RD) leads.

Figure D–3 demonstrates use of a breakout box for a specific application. Recall that a DMx controller will not operate unless the modem asserts Data Set Ready (Circuit CC – Pin 6). However, many modems negate DSR when analog loopback is enabled. To get around this problem, a breakout box can be set up as shown in Figure D–3. Note that Data Terminal Ready (Circuit CD – Pin 20), that is asserted by the DMx whenever it is enabled, is connected to Data Set Ready. Furthermore, the DSR lead from the modem is broken by an open switch. Hence, DSR, as seen by the controller, is asserted whenever DTR is asserted (that is, whenever the DMx is ON). Since many modems do not process control signals when operating in a loopback mode, several other patches may also be necessary. In Figure D–3, Request to Send (Circuit CA – Pin 4), that is asserted by the controller when it wants to transmit, is fed back to the device as Clear to Send (Circuit CB – Pin 5) and Data Carrier Detect (Circuit CF – Pin 8). The CTS and DCD leads from the modem must also be broken by open switches. This arrangement will allow an analog loopback test to be run with a modem that either negates Data Set Ready or does not process control signal when analog loopback is enabled.

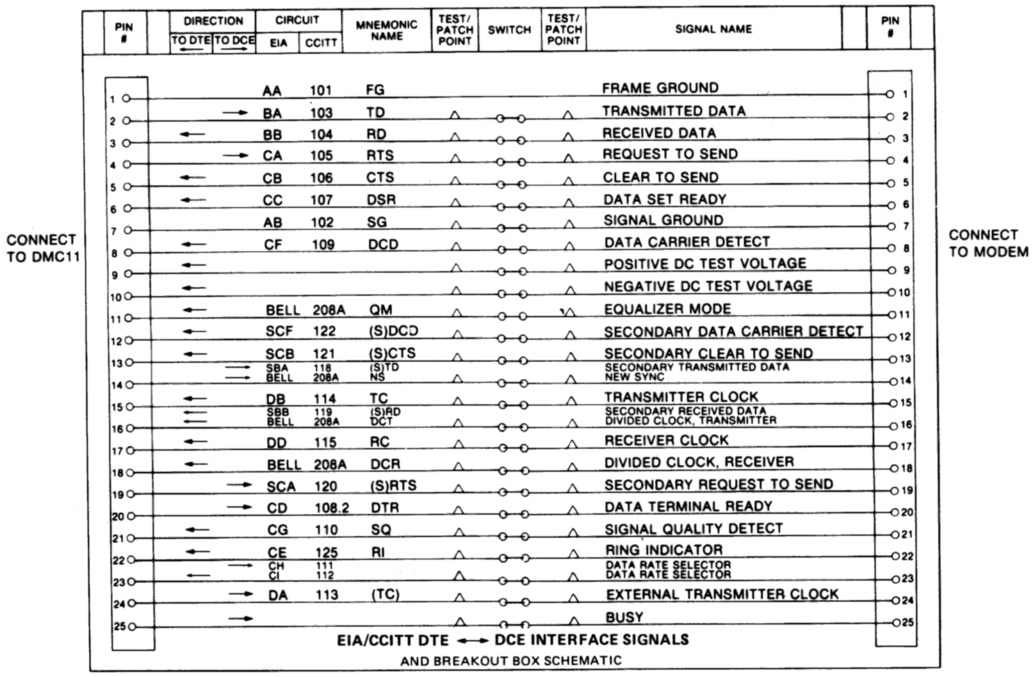
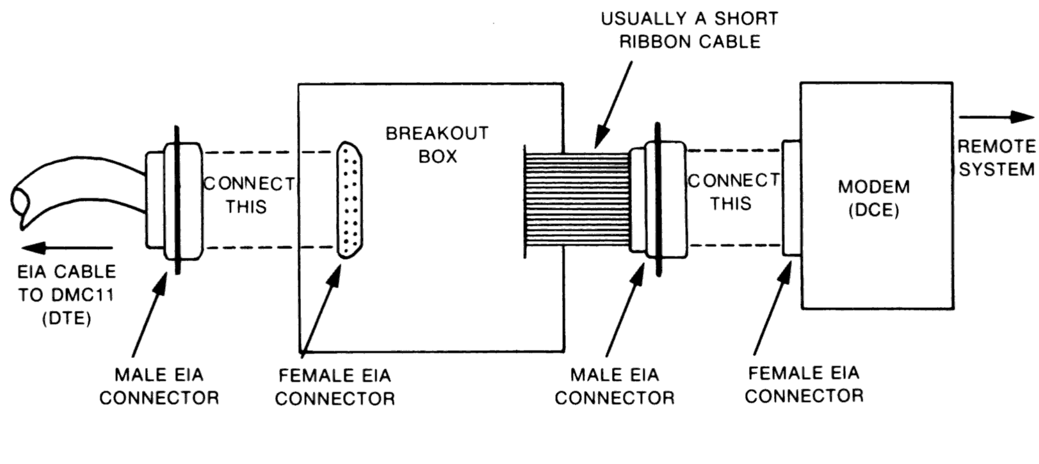


Figure D–1: Breakout Box Passing All EIA Signals



Q-MK-00008-00

Figure D–2: Connecting a Breakout Box

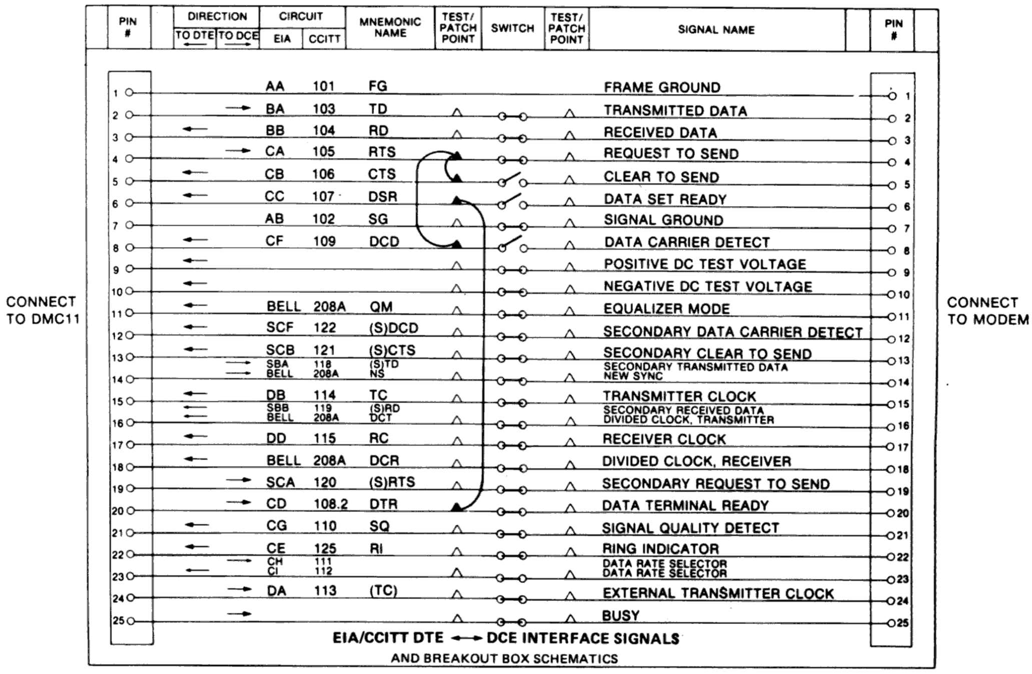


Figure D–3: Breakout Box Connections for Analog Loop Test

Appendix E
DTS/DTR Utility Program

The Data Test Sender (DTS) and the Data Test Receiver (DTR) are the DECnet/E transmitter and receiver test programs. The object libraries and command files to build DTS and DTR are included on your DECnet/E distribution kit and are used during the system generation procedure. Appendix A (Section A.6) lists the DTS/DTR command files, also supplied on the distribution kit, that should be used to test connections to adjacent Phase II nodes.

E.1 Types of Tests

There are four basic tests provided by DTS and DTR.

- Connect Tests
- Data Tests
- Disconnect Tests
- Interrupt Tests

Each test is divided into a set of subtests. The tests and subtests are described in the following sections.

E.1.1 Connect Tests

Connect tests are designed to test the ability of the network software to process connect, connect accept, and connect reject requests – with and without optional user data. Connect tests that the user can perform are as follows:

- Connect reject without user data
- Connect accept without user data
- Connect reject with 16 bytes of standard user data
- Connect accept with 16 bytes of standard user data
- Connect reject with received user data used as reject user data
- Connect accept with received user data used as accept user data

E.1.2 Data Tests

Data tests provide a full range of test capabilities from the very simple data sink operation through data integrity checking. Data tests that the user can perform are as follows:

- **Sink Test.** DTR ignores all data received. No sequence of content validation is performed.
- **Sequence Test.** Data messages transmitted by DTS or DTR include a 4-byte sequence number. If a message is received out of sequence, DTR aborts the logical link and the test.
- **Pattern Test.** Data messages transmitted to DTR have both a sequence number and a standard data pattern. If either the sequence number or the received data does not match the expected data, DTR aborts the logical link and the test.
- **Echo Test.** Data messages received by DTR are transmitted back to DTS. No sequence or data validity checking is done by either DTR or DTS.

E.1.3 Disconnect Tests

Disconnect tests are designed to determine whether DTS can detect the difference between disconnect and abort sequences generated by DTR as well as receive the proper optional user data. Disconnect tests that can be performed by the user are as follows:

- Disconnect without user data
- Abort without user data
- Disconnect with 16 bytes of standard user data
- Abort with 16 bytes of standard user data
- Disconnect with received connect user data as disconnect user data
- Abort with received connect user data as abort user data

E.1.4 Interrupt Tests

Interrupt tests provide a full range of test capabilities from very simple data sink operations through data integrity checking. Interrupt tests that the user can perform are as follows:

- **Sink Test.** DTR ignores all interrupt data received. No sequence or content validation is performed.
- **Sequence Test.** Interrupt messages transmitted by DTS to DTR include a 4-byte sequence number. If a message is received out of sequence, DTR aborts the logical link and the test.

- **Pattern Test.** Interrupt messages transmitted to DTR have both sequence number and a standard data pattern. If either the sequence number or the received data does not match the expected data, DTR aborts the logical link and the test.
- **Echo Test.** Interrupt messages received by DTR are transmitted back to DTS. No sequence or data validity checking is done by either DTR or DTS.

E.2 Operational Characteristics

DTR functions as a slave to DTS. DTS initiates each test by issuing a connect request to DTR. Parameter information pertinent to the type of test requested is passed by DTS to DTR in the optional data of the connect request. DTS has a user interface that enables the user to specify the test to be performed. Sufficient parameters are available to allow for a variety of tests, including test duration, buffer size, and buffering level.

The version of DTR supplied on the DECnet/E distribution kit supports a maximum of three logical links, thus allowing three concurrent tests. DTS can handle only a single test and logical link, but the user can invoke multiple copies of DTS.

The DTS command syntax allows two types of buffer level parameters: DEPTH and FCOUNT. DEPTH is used by DTS only and FCOUNT is used by DTR only. The DEPTH parameter specifies the transmit buffering level for DTS. This is the number of transmission requests that DTS attempts to keep outstanding to the network. In an echo test, the DEPTH parameter also specifies the receive buffer level for DTS. The FCOUNT parameter specifies the receive buffering level for DTR.

Both DTS and DTR have 512 bytes of buffer space, built into each task, at the end of each task image. To increase the buffer space of either task, the user must edit the BASIC program and recompile. DTS requires at least (DEPTH*SIZE) bytes of buffer space for a data test, and ((DEPTH+1)*SIZE) bytes for an echo test. DTS requires (FCOUNT*SIZE) bytes of buffer space for each data test.

E.3 DTS Command Syntax

The syntax of DTS commands is very "free form" in nature. Commands are composed of keywords and parameters, separated by whitespace (i.e., tabs or spaces) and/or special characters. Colons (":") and equal signs ("=") are interchangeable and are used to associate parameters with keywords. Parameters take two forms: numeric or text.

A command must be completed in a single line. Keywords can begin with or without slashes (as the user sees fit), and the name of the test to be run must

be specified before any options. Thus, the following are all valid (and synonymous) commands:

```
DATA ECHO  /SIZE=256  /MINUTES=2  /FLOW=SEGMENT/FCOUNT=1

/DATA/TYPE=ECHO/FLOW=SEGMENT/FCOUNT=1/SIZE=256/MINUTES=2

DATA TYP:ECHO FLO:SEG FC:1 SIZE:256 MIN:2
DATA/SIZ=256  ECHO  SEG/FC=1  MIN=2
```

Formats for DTS commands are described below. The following conventions apply to the presentation and use of these commands:

Parentheses ( )	Indicate an optional parameter.
Braces { }	Indicate a choice of parameters or keywords.
UPPERCASE	Indicates the actual keyword or parameter text that appears in the call.
lowercase italic	Indicates a parameter that is replaced with an actual value when invoked.

E.3.1 Specifying the DTS Node

$\left\{ \begin{array}{l} \text{(NODE} = \textit{nodename}) \\ (\textit{nodename}\text{::}) \end{array} \right\}$ (/UID = *uid*) (/PASSWORD = *pwd*) (/ACCOUNT = *act*) (/PRINT)

nodename	Specifies the name of the node where DTR resides. Default is the local node.
uic	Specifies the user ID, project-programmer number (PPN), or user identification code (UIC) for access to the remote node.
pwd	Specifies the password for access to the remote node.
act	Specifies the billing account for access to the remote node.

NOTE

The /PRINT option is used by DECnet/E DTS to either display or suppress message output to the local console device. Messages include both test results and error messages.

E.3.2 Connect Test

CONNECT *type* $\left\{ \begin{array}{l} \text{/NORETURN} \\ \text{/RETURN} = \textit{data} \end{array} \right\}$

type Type of connect test:

ACCEPT	Connect accept test
REJECT	Connect reject test

data Type of user data returned:

STANDARD	Standard user data
RECEIVED	Return connect user data

E.3.3 Disconnect Test

DISCONNECT *type* { /NORETURN | /RETURN = *data* }

type Type of disconnect test:

SYNCHRONOUS	Synchronous disconnect test
ABORT	Abort test

data Type of user data returned:

STANDARD	Standard user data
RECEIVED	Return connect user data

E.3.4 Data Test

DATA *type* { /SECONDS = *nn* | /MINUTES = *nn* } /SIZE = *mm* /DEPTH = *bb* { /NOFLOW | /FLOW = *flt* } /FCOUNT = *ff*

type Type of data test:

SINK	Sink test
SEQUENCE	Sequence test
PATTERN	Pattern test
ECHO	Echo test

nn Test duration.

mm Data message length, in bytes. Must be greater than 0 for for a sink or echo test, greater than 4 for a sequence test, and greater than 5 for a pattern test. The maximum value is 1024 bytes.

bb DTS transmit buffering level, in the range of 1 to 16.

flt Flow control type:

SEGMENT	Segment flow control
MESSAGE	Message flow control

ff Number of segments/messages requested if /FLOW was selected.

E.3.5 Interrupt Test

INTERRUPT *type* { /SECONDS = *nn* | /MINUTES = *nn* } /SIZE = *mm*

type Type of interrupt test:

SINK	Sink Test
SEQUENCE	Sequence test
PATTERN	Pattern test
ECHO	Echo test

nn Test duration

mm Data message length, in bytes. Must be greater than 0 for a sink or echo test, greater than 4 for a sequence test, and greater than 5 for a pattern test. The maximum value is 16 bytes.

INDEX

E

F

H

I

L

M

N

O

P

R

S

T

U

V

X

DECnet/E
Network Installation Guide
AA-K714A-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)__________

Name__________ Date__________

Organization__________

Street__________

City__________ State__________ Zip Code or Country__________

— — — — —Do Not Tear - Fold Here and Tape — — — — — — — — — — — — — — — — —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE DOCUMENTATION
1925 ANDOVER STREET TW/E07
TEWKSBURY, MASSACHUSETTS 01876

— — — — Do Not Tear - Fold Here and Tape — — — — — — — — — — — — — — — —

Cut Along Dotted Line